# Towards Predictive Rendering in Virtual Reality

Dissertation

zur Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

**M.Sc. Jan Meseth**

aus München

Bonn, Oktober 2006

Universität Bonn,
Institut für Informatik II
Römerstraße 164, 53117 Bonn

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms Universität Bonn.



| | |
|---|---|
| Dekan: | Prof. Dr. Armin B. Cremers |
| 1. Referent: | Prof. Dr. Reinhard Klein |
| 2. Referent: | Prof. Dr. Stefan Müller |
| Tag der Promotion: | 08.11.2007 |
| Erscheinungsjahr: | 2008 |

# Acknowledgments

Creating this thesis required a tremendous amount of work which I could not have accomplished without the support of many people. Acknowledging these people here is clearly an insufficient expression of my heartfelt gratitude to them. Yet, I hope that all of them know, how deeply I appreciate what they have done for me.

The person most important for this thesis is my adviser Prof. Dr. Reinhard Klein, a frequent source of inspiration with a sheerly infinite number of great ideas. He was the one truly introducing me to scientific work, transferring such a huge amount of knowledge to me by providing novel points of view and helpful criticism, and pushing me to previously unimagined performance.

Of similar importance to this work are my former and current colleagues from the Computer Graphics group. During the years of partially collaborative work, the discussions of scientific and profane problems, many of them became much more than ordinary colleagues. From a professional point of view, I especially thank Gero Müller, Mirko Sattler, Ralf Sarlette, Michael Guthe, and Ákos Balázs for preparing joint publications with me. From a personal point of view, my dearest thanks go to my Hungarian friends Marcin and Dominik Novotni, Ferenc Kahlesz, and Ákos Balázs, and to Patrick Degener, Gerhard Bendels, Gero Müller, and Simone von Neffe.

Additionally, credits belong to my diploma students André Nicoll, Florian Röder and Georg Breitsprecher for conducting excellent work with an amount of commitment significantly exceeding the necessary requirements.

I thank all RealReflect project partners for the good and successful collaboration. They have often been a significant source of inspiration and qualified criticism.

I also want to acknowledge the various sources of data used in this work. Several of the texture images are from Paul Bourke's texture database. The Mercedes C-class model, the Golf model, the scanned man model, the Max Planck bust model and the Welfenschloss model were generously provided by the DaimlerChrysler AG, Volkswagen AG, the Virtual Try-On research project, the Max-Planck Institute for Informatics, and the Institute for Cartography and Geoinformatics at Leibniz University Hannover. Additionally, I want to thank Paul Debevec for making available his HDR environments, and Rafał Mantiuk for providing and updating his HDR VDP code.

While this thesis would not have been possible without the mostly technical support of my adviser and my colleagues, the personal support of family and friends was of probably even greater importance. It is impossible for me to express the real importance of my parents and siblings for starting, conducting, and finishing this thesis. They have always been an unquestioned source of love and support, acting as a safe harbor during all the stormy times in my life. They are irreplaceable for me.

Also, I deeply thank my friends in Bonn for sharing many great evenings, weekends, and even holidays, accompanying me in times of joy and sadness, giving me strength and hope to tackle the never ending cruelties of life. In addition, I thank my friends from home for never forgetting me, caring about and loving me, supporting me, warmly welcoming me after months of absence without any reserve.

I thank Birte Wehnsen for three wonderful years.

Thanks also go to the various football teams who played so many great or horrible games with me, especially for withstanding and accepting my sometimes overmotivated style of playing. I also always loved to go to my choir, where I was further educated in the art of music and where I experienced very many nice practice hours, practice weekends, and beautiful concerts.

# Abstract

The strive for generating predictive images, i.e., images representing radiometrically correct renditions of reality, has been a longstanding problem in computer graphics. The exactness of such images is extremely important for Virtual Reality applications like Virtual Prototyping, where users need to make decisions impacting large investments based on the simulated images.

Unfortunately, generation of predictive imagery is still an unsolved problem due to manifold reasons, especially if real-time restrictions apply. First, existing scenes used for rendering are not modeled accurately enough to create predictive images. Second, even with huge computational efforts existing rendering algorithms are not able to produce radiometrically correct images. Third, current display devices need to convert rendered images into some low-dimensional color space, which prohibits display of radiometrically correct images.

Overcoming these limitations is the focus of current state-of-the-art research. This thesis also contributes to this task. First, it briefly introduces the necessary background and identifies the steps required for real-time predictive image generation. Then, existing techniques targeting these steps are presented and their limitations are pointed out. To solve some of the remaining problems, novel techniques are proposed. They cover various steps in the predictive image generation process, ranging from accurate scene modeling over efficient data representation to high-quality, real-time rendering.

A special focus of this thesis lays on real-time generation of predictive images using bidirectional texture functions (BTFs), i.e., very accurate representations for spatially varying surface materials. The techniques proposed by this thesis enable efficient handling of BTFs by compressing the huge amount of data contained in this material representation, applying them to geometric surfaces using texture and BTF synthesis techniques, and rendering BTF covered objects in real-time. Further approaches proposed in this thesis target inclusion of real-time global illumination effects or more efficient rendering using novel level-of-detail representations for geometric objects. Finally, this thesis assesses the rendering quality achievable with BTF materials, indicating a significant increase in realism but also confirming the remainder of problems to be solved to achieve truly predictive image generation.

# Contents

# Part I

# Introduction

# Chapter 1

# Definitions

## 1.1 Virtual Reality

The term *Virtual Reality* (VR) is probably known to most people today. Everyone has some ideas of what this term implies, ranging from threatening scenarios where the user is caught in some virtual world as in the movies *Tron* by Steven Lisberger or *The Matrix* by the Wachowski brothers, over strange, discerning scenarios where people live and die in hyper-developed Internet scenarios as in the *Neuromancer* novel from William Gibson, to helpful, entertaining scenarios where people use holographic rooms for their own comfort and relaxation on a daily basis as in the *Star Trek* series by Gene Roddenberry. These associativities are typically established by books and movies playing in the future and do not represent the current state of Virtual Reality – although similar scenarios might become reality in the future.

From a current, unemotional point of view VR can be defined as a *"... computer system used to create an artificial world in which the user has the impression of being in that world and the ability to navigate through the world and manipulate objects in the world."* [339] This definition describes many important aspects of VR whereas most are accepted by most people from the area of VR.

Naturally, VR deals with artificial worlds that are presented to some users and which the users can interact with. As described by the definition, these worlds typically mimic reality, which includes possibilities for interaction with and manipulation of objects from this world in a natural way. Typically artificial worlds are presented to the user by specialized display devices, because – according to Bricken [48] – *"... the primary defining characteristic of VR is inclusion, being surrounded by an environment. VR places the participant inside information."* Therefore, display devices usually feature stereo imaging, high-resolution imaging, and often consist of multi-side projection systems where the user can literally stand within the projected virtual world.

The most important aspect of VR was formulated by the computer graphics pioneer Ivan Sutherland in 1965 already when describing his idea of an ideal display: *"Don't think of that thing as a screen, think of it as a window, a window through which one looks into a virtual world. The challenge to computer graphics is to make that virtual world look real, sound real, move and respond to interaction in real time, and even feel real"* [50]. According to him, the task of VR is simulation of a virtual world in such a realistic way that the user gains the impression of being inside a real world, which is called *immersion*. To achieve this immersion multi-modal interaction including all senses (sight, hearing, touch, taste, smell) should be provided in a real-time manner and with such accuracy that the user cannot distinguish between actual and virtual reality.

Of course, such a perfect VR system does not exist yet. Nevertheless, VR is successfully employed in a large number of application areas already, which can roughly be classified into four groups: virtual prototyping, training and simulation, edutainment, and collaborative work.

Virtual Prototyping has become an important aspect of product design and its importance is growing steadily. It was first employed in architectural application scenarios (architecture, urban management, and interior architecture to some small extent). Today it has found its way into several industrial areas like the automotive, ship, and aircraft industry and will soon achieve big importance in most production industries. The key idea is to replace all types of physically existing models during product conception, styling, design, and analysis by digital models which can be built faster and at lower costs, which can be reused at no costs (e.g., during crash tests in the automotive industry), which can be analyzed in hypothetical environments (e.g., arbitrary lighting conditions), and which enable automatic processing (e.g., flow simulations in the aircraft industry). VR especially supports detecting bad decisions earlier in the design process, and helps to reduce the amount of time and money spent per variant.

A related yet different application area is training and simulation. VR is typically employed to simulate and train dangerous situations like surgeries, military operations, or problems during flights in airplanes, rockets or spacecrafts. Other applications simulate assembly and disassembly of complex products like cars with the target to optimize the costs or ergonomic aspects related to these tasks.

An application area related to training and simulation is edutainment, which is a mixture of the words education and entertainment. Following the principle that students learn more the more senses they use when learning, VR represents an optimal medium for learning. The ability to interact with environments in a natural manner makes learning typically more joyful. This principle is often used in computer games as well, which can safely be regarded VR software since they feature realistic real-time graphics and sound, and often even simple tactile interfaces. A less interactive yet often more immer-

sive application area of VR are 3D movie theaters which show highly realistic images and sound, and which may feature input for additional senses like smell or touch.

Last but not least, collaborative work is an important application area which allows people at remote places to work together in a natural way. Existing applications range from simple video conferencing software over monitoring and control systems to methods for remote, collaborative editing and design.

Whereas a large number of VR applications exist already, the last years have shown that VR had less success than expected during the initial hype in the 1980s. Reasons for this development were high costs for and bad handling of specific devices and especially lack of presentation quality due to missing hardware and software solutions for application areas that potentially might profit from VR. This thesis shall help to resolve some of these problems by presenting software developments and solutions for specific application areas.

## 1.2 Predictive Rendering

The second term fundamental to this thesis is *Predictive Rendering*. Whereas most people from the computer graphics area agree that the term *Rendering* describes the process of generating a 2D image from a scene description, the meaning of the term *Predictive Rendering* is not concisely agreed on so far. While Dmitriev et al. define a predictive rendering algorithm as a method for *"Synthesis of realistic images which predicts the appearance of the real world . . ."* [110], Ulbricht et al. make a stronger statement. They define predictive rendering algorithms to be methods that *". . . yield results that do not only look real, but are also radiometrically correct renditions of nature, i.e. which are accurate predictions of what a real scene would look like under given lighting conditions.*" [528]

Both statements include the idea of predicting, which refers to making a statement about a hypothetical state or event. In computer graphics such situations typically occur when statements concerning a virtual model, i.e., a model that exists in digital form only, have to be made. Therefore, predictive rendering algorithms need to have the ability to produce correct results without having ground truth comparisons from the real world during rendering time.

The second statement additionally requires predictive rendering algorithms to produce radiometrically correct results. This definition differentiates this class of algorithms from the photorealistic rendering methods as defined by Ferwerda [141] and Pharr and Humphreys [412]. They state that the goal of photorealistic rendering algorithms is to *". . . create an image which is indistinguishable from a photograph of the*

*same scene.*" [412]. According to this definition, photorealism includes interactions of light with some kind of camera (possibly the human eye) and thus requires photometric correctness instead of radiometric correctness. Yet, photometric correctness leads to the problem that images perceived as photorealistic by one observer may be perceived as not being photorealistic by others. This does not apply to radiometrically correct images. Therefore, predictive rendering algorithms are a subclass of photorealistic algorithms since every radiometrically correct image is photometrically correct as well.

Whereas achieving photorealism during rendering is a challenging task, achieving predictive rendering is even more complex. The required accuracy of photorealistic imaging methods is typically limited by the perception capabilities of the human visual system. For predictive rendering, such a bound does not exist. Therefore, a perfect simulation of the interaction of light and matter is required to achieve true predictability. The class of rendering methods that try to reach this goal are called *Physically Based Rendering* algorithms.

Of course, such a perfect simulation is not achievable, at least in the near future. A theoretical limitation arises from the problem that many aspects of light-matter interaction are not (fully) understood today and will probably remain unknown in the future. A more practical limitation stems from the computational efforts required to simulate all known effects in scenes with practically relevant size and complexity.

Therefore, existing physically based rendering methods make simplifying assumptions. First, geometric optics are applied, omitting effects related to the representation of light as waves or particles (e.g., diffraction, dispersion, polarization, and fluorescence). Although Shirley assumes that *"... for at least the next couple of decades, a lack of polarization or physical optics effects will not be the biggest limitation of rendering codes"* [477] such effects need to be incorporated to achieve predictive results (see, e.g., [579] or [47]). Second, the spectrum of wavelengths is discretized into a fixed number of spectral bands. Interactions between these bands are often ignored.

Efficiently removing these limitations is of course quite challenging. Yet, not only rendering has to be improved. Additionally, new modeling paradigms have to be determined that allow specification of parameters influencing predictive image generation (i.e., material or light source properties). Clearly, modeled lights and materials have to mirror real-world counterparts – although those need not necessarily exist.

Due to their focus on physical correctness, predictive rendering is limited to application scenarios which do not require significant artistic freedom. Especially, physically impossible effects cannot be modeled. Therefore, such algorithms will typically not be used for movie creation or computer games. Nevertheless, the market potential of these approaches if high due to their relevance in industrial prototyping, architecture, and many other application areas.

**Fig. 1.1: Comparison of rendering quality achievable with current VR system (left) and existing physically based renderer including measured light and materials (right).**

## 1.3 Predictive Rendering in Virtual Reality

The definition of VR onto which this thesis bases includes the strive for perfect simulation of sensorial stimuli for all human senses. Therefore, utilization of predictive rendering in VR applications appears to be an obvious step towards reaching the definition's goal. Nevertheless, despite the clear advantages of having physically correct images such a combination is not available today.

Existing VR applications employ relatively simple rendering methods that cannot guarantee any physical realism. Typically only local illumination models or radiosity solutions are employed, which are supported by real-time rendering APIs and which are accelerated by graphics hardware. While the geometry to be rendered is typically relatively accurate, the materials and light sources usually feature a poor degree of realism. To improve the spatial feeling and to add more realism, often simple shadowing algorithms are employed.

The left image in Figure 1.1 shows the image quality achievable with such an approach. Materials were modeled in a standard way using a combination of a Phong BRDF [413] and diffuse textures. One point light was placed in the driver's position to brighten the scene, a second directional light simulated the sun. At a first glance, the image gives a very realistic impression. Yet, in comparison to the right image, which was computed with a physically based rendering algorithm including measured light and materials, two main deficiencies become obvious. First, the overall distribution of light is completely different. One of the reasons is the introduction of the point light source to achieve a satisfying level of brightness in the region of interest which results

in overly bright areas. In contrast, other regions like the footwell are too dark because indirect light is omitted (which is due both to the local illumination model and the omission of sunlight reflected from the cloudy sky). Finally, the sharp shadows clearly contrast with most scenarios from reality. Second, the materials do not look the way they are supposed to look, which can especially be seen in the enlarged views of the dashboard. Despite of using a diffuse texture, the appearance of the material in the left image is flat, which makes it very difficult to recognize the material as one of the coined synthetic materials typically used in cars. The degree of realism is significantly improved in the right image where the impression of a material with height variance becomes apparent. Another problem contributing to the odd appearance of materials is the difficulty of correctly modeling colors and BRDFs by simplistic models. E.g., the colors from the lacquered wood placed on the center console significantly differ, with the right image being much closer to reality than the left one.

These two reasons currently prohibit the use of VR in scenarios that require an (almost) exact reproduction or prediction of a product's appearance. Examples of such application areas include interior design of buildings, cars, ships, trains, and airplanes, design of products with complex material properties (e.g., garments, jewelry) or light interactions (e.g., headlights), safety-relevant analysis of products, and many more.

Unfortunately, combinations of VR systems with predictive rendering algorithms are difficult to implement due to conflicting requirements and missing technology. While the strive for immersion in VR systems clearly includes the necessity of realistic image synthesis, experience determined that the need for interactivity is stronger. Images need to be generated at real-time frame rates in order to give the user an impression of being immersed even if they are generated in some crude way. Most existing methods for physically based rendering cannot fulfill real-time requirements at the moment but instead require seconds or minutes for a single image.

Another problem prohibiting predictive rendering stems from inadequate modeling of VR scenes. To achieve optimal rendering quality, among other things optimal geometry, light and material representations have to be employed and materials have to be applied to geometries in a fashion mimicking reality. The models and methods currently used are far from being optimal.

A third limiting factor is display technology: Even if the rendering algorithms produce radiometrically correct results, they have to be presented to the viewers in some way. Existing VR displays typically lack high display resolution, high dynamic range, correct reproduction of color, and many other things. To cope with these problems specific software solutions have to be employed, but without introduction of better display technology the ultimate goal of predictive rendering in VR will remain unreachable.

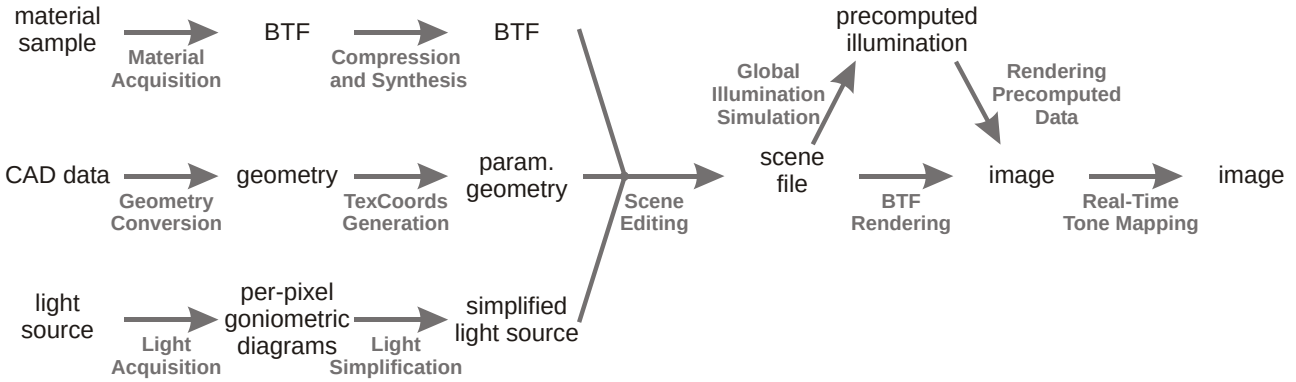



**Fig. 1.2: RealReflect data preprocessing and image generation pipeline.**

## 1.4 Predictive Image Generation Pipeline

To resolve or at least ease the abovementioned problems limiting the usefulness of predictive rendering in VR, a series of fine-tuned steps has to be applied. Starting with accurate modeling, they also need to cover real-time, physically-based (or at least photorealistic) rendering and methods for best-possible display of generated images on existing display devices. Such a series of steps was first proposed by the RealReflect [435, 272] research project whose goal was the improvement of rendering quality in VR applications to a level where VR would become useful in application areas like car interior design or interior architecture. The project had to tackle all challenges occurring when introducing predictive rendering into VR systems (i.e., predictive rendering quality on existing display devices at real-time frame rates for large scenes). The series of necessary, conceptual steps was implemented as a pipeline which is depicted as a flow diagram in Figure 1.2.

The first steps are devoted to accurate modeling of scenes. Geometric models are derived from CAD modeling tools. Digital representations of materials and light sources are specified by measuring the most relevant characteristics of existing samples instead of modeling them explicitly. In order to generate predictive images the rendering algorithms include as many physically-based effects as possible. Two different ways of rendering are supported: one requiring precomputation and one working without. Naturally, a tradeoff between precomputation time and storage, rendering speed, and rendering quality has to be accepted. To improve rendering performance, level of detail and occlusion culling algorithms are employed. To minimize additional reductions of the final rendering quality introduced by existing, imperfect display devices, rendered image are tone-mapped before displaying them – taking into account the characteristics of both the display system and the human. In the following the individual stages shown in Figure 1.2 are described in more detail.

**Light Acquisition** The light acquisition stage has the task of digitizing radiance distribution data of existing light sources (a light source being a combination of light bulbs and geometry including mirrors). Accurate measurement of this data requires acquisition of near-field photometric data, e.g., by capturing the *Luminance Field* [8] – a 4D function specifying the distribution of radiation by a goniometric diagram per surface point. Unfortunately such accurate data cannot be employed in real-time rendering methods but for highly realistic precomputations only. To support as accurate as possible real-time rendering, far-field photometric data (which assumes the light source to be a point light and models the radiance distribution by a single goniometric diagram) and distant lighting environments are derived.

**Material Acquisition** Like the light acquisition stage, material acquisition has the task of digitizing reflectance properties of existing materials. A compromise between accuracy and required acquisition time was made by deciding to acquire Bidirectional Texture Functions (BTFs) [371] – 6D functions specifying outgoing radiance for every view direction, light direction, and surface position. Due to the large amount of data acquired per material (typically several GBs) data compression needs to be applied to enable efficient processing.

Unlike light acquisition, which assumes a light source of fixed extent, materials need to have arbitrary extent in order to cover arbitrarily large object surfaces. To enlarge the fixed extent of acquired samples (typically 10 cm $\times$ 10 cm) texture synthesis approaches are required. Obviously, these synthesis methods need to preserve each material's appearance in order to enable highly realistic rendering.

**Geometry Preparation** Geometry is handled in a different way than material and light: It is not measured but modeled since this reflects the standard way in Virtual Prototyping, which heavily relies on highly accurate, modeled CAD geometry already. Yet, some processing of the existing data is necessary since requirements for CAD data are typically different than for data used for efficient rendering. Therefore, inconsistent triangulations have to be repaired, inconsistently oriented patches have to be oriented in a consistent way, problems with bad normals have to be fixed, and most importantly texture coordinates for applying materials have to be computed.

**Scene Editing** After digitizing and preprocessing all required data, materials need to be assigned to geometry, and objects and light sources have to be placed in scenes. Since real-time rendering typically requires reducing the complexity of rendered scenes, Level of Detail (LOD) representations of the objects are employed to adapt the detail of

displayed parts to the display resolution. Additionally, occlusion culling methods avoid rendering of parts invisible in the final image. Within the scene editing stage, necessary precomputations for these approaches are performed.

**Rendering** The task of transforming the scene description into an image is denoted as rendering. Here, two separate ways for rendering are employed. The first approach precomputes (parts of) the global illumination solution for the current scene in an offline process, and stores relevant data in appropriate data structures. The precomputation step aims at achieving as accurate as possible results by employing physically-based rendering algorithms. Due to the large amount of data stored for scenes with practically relevant size, data compression is applied. Finally, the precomputed data is evaluated in real-time and shown to the user, giving a very accurate approximation of predictive rendering in real-time.

The second approach sacrifices accuracy in the rendered images by omitting the precomputation step and employing combinations of local illumination models for simplified light sources (goniometric point lights or directional lights, and environment maps encoding distant lighting) and shadow computation algorithms.

Both approaches need to avoid reduction of rendering quality wherever possible (i.e., the original dynamic range of light, material and colors should be preserved) but need to make compromises to enable real-time rendering (i.e., discretization of the light spectrum to tristimulus values).

**Tone-Mapping** The final step adjusts the rendered images in real-time such that they can be shown on existing display devices with optimal quality. Tone-mapping methods have to take into account the properties of display devices (e.g., limited dynamic range and color gamut) and should include the characteristics of the human visual system (e.g., adaptation, visual acuity, and glare). In addition, they allow simulation of certain deficiencies of the human visual system, providing insight into potential problems for affected people.

Despite the improvements in rendering quality that can be expected due to the novel approach, the results will clearly not reach the level of predictive rendering since too many effects (e.g., spectral rendering, polarization, diffraction, etc.) cannot be taken into account due to lack of existing methods and limitations of current hardware. To evaluate the success of the novel approach, evaluations concerning the improvement of rendering quality and the remaining problems were made, both for individual stages and the complete setup.

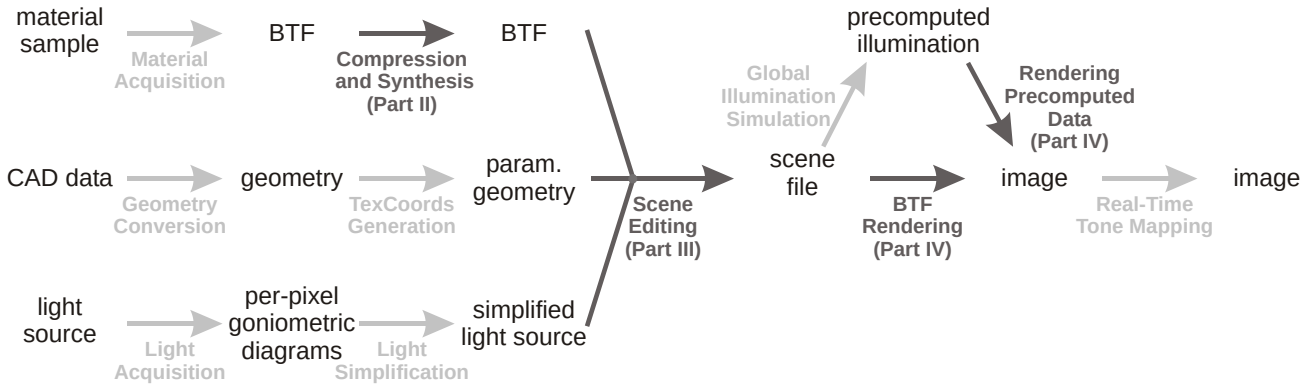



**Fig. 1.3: Own contributions to the predictive image generation pipeline. Dark-colored transitions mark processes to which this thesis provides own contributions. High-level overviews of existing work related to the light-colored stages complete the description.**

## 1.5 Scope of Thesis

This thesis emanates from research done within the RealReflect project. Therefore, it describes results that contribute to the overall workflow without contributing to all parts (cf. Figure 1.3). It lays a focus on efficient and high-quality rendering of scenes containing objects covered with complex material properties. Thus, aspects like light acquisition, geometry preparation and tone mapping are covered only briefly in order to give an understanding of the bigger picture. More details concerning these topics can be found in the course notes from Ian Ashdown [9] and the book of Michael Goesele [164], which cover light acquisition methods, the various existing books on geometry processing, the book on high-dynamic range imaging and tone mapping by Reinhard et al. [436], and the tutorial on HDR techniques by Myszkowski et al. [374].

The thesis is composed of six parts. This first part is devoted to introducing especially relevant terms and related problems. Additionally, it is supposed to provide background knowledge necessary for understanding the scope of the thesis.

The following two parts concentrate on modeling scenes suitable for efficient predictive rendering algorithms. Part II concentrates on the importance of accurate material representations for predictive rendering. An overview over material representations is given in Chapter 2. It covers approaches for modeling such data as well as showing ways for measuring reflectance data, including methods for data compression and sample enlargement. A high-quality method for compression of BTF data is described in Chapter 3. In Chapter 4 a method for sample enlargement of near-regular materials based on sample-based texture synthesis is presented. Additionally, experiments with BTF synthesis techniques are summarized.

Next, Part III details the significance of level of detail methods for efficient predictive rendering. Again, the initial Chapter 5 describes state of the art methods, with a special focus on polygonal and parametric surface representations, including influences of surface materials. Chapter 6 then describes a method for efficient level of detail handling of surfaces consisting of trimmed NURBS patches, which is of extreme importance for CAD models from automotive industry. Following, Chapter 7 provides details on an approach for extreme simplification of objects consisting of arbitrary surface representations and covered with complex materials.

After describing approaches for preparing scenes viable for efficient predictive rendering, Part IV of this thesis presents algorithms for real-time physically-based rendering. As the previous parts, it starts with a survey of existing methods for real-time physically based rendering in Chapter 8, including a brief overview of principal approaches for physically based rendering. Then, methods for real-time rendering of scenes containing objects covered with compressed BTF materials and measured light sources are presented in Chapter 9. Possible light sources include environmental lighting and measured far-field photometric data (i.e., goniometric point lights). In the following chapters, the achieved rendering quality is improved by adding global illumination effects. Chapter 10 describes a method based on precomputation and real-time evaluation of Surface Light Fields (SLFs), and documents a method based on Precomputed Radiance Transfer (PRT) which achieves a higher degree of flexibility than SLF rendering. Although both approaches show impressive visualization quality, they do not support combinations of high-frequency lighting and materials due to the required storage requirements for precomputed data. Chapter 11 presents an approach for interactive computation of reflections in highly reflective surfaces like mirrors. This represents a method for compensation for the limitations of SLF and PRT rendering but obviously combines nicely with rendering methods based on local illumination models as well.

The following Part V deals with evaluations of the rendering quality achieved with accurate materials and lights. First, Chapter 12 gives an overview of related methods for evaluation of rendering quality, then Chapter 13 provides an explicit evaluation for measured BTFs and measured environmental lighting.

The final Part VI contains two chapters. In Chapter 14 the results achieved within this thesis are described and their implications for real-time predictive rendering applications are shown. Obviously many things remain to be done to achieve truly predictive rendering in VR. Therefore this thesis ends with an outline of necessary improvements to be implemented in the future in Chapter 15. Many of these points will hopefully guide research in the near or further future since the goal of predictive rendering in VR appears to be very desirable and promising.

# Part II

# Material Representations

# Chapter 2

# State of the Art

In the previous part of this thesis, predictive rendering was defined as the process of generating images of a scene which represent the amount and spectral distribution of light arriving at an observer's imaging organ or device. As mentioned already, the ability to produce such images clearly depends on the accuracy of the modeled scene. Especially modeling of light-matter interaction plays a crucial role. While such interactions occur at all geometric scales, in this part of the thesis a focus is layed on the influence of small-scale geometric detail, which is too small to be modeled explicitly and which is commonly considered to be a *material* property. Although materials have many properties relevant to realistic reproduction in VR systems, this thesis concentrates on reflectance properties, i.e., those properties that influence the reflection behavior of materials.

Figure 2.1 sketches the general case of light matter interaction. A photon with wavelength $\lambda_i$ hits an object surface at time $t_i$ at point $\mathbf{x}_i$ from local direction $(\theta_i, \phi_i)$, then travels through the object, and later leaves its surface at time $t_r$ at position $\mathbf{x}_r$ into direction $(\theta_r, \phi_r)$. Due to fluorescence, the wavelength $\lambda_r$ of the emitted photon may be different from the wavelength of the incoming light. This model is based on a very

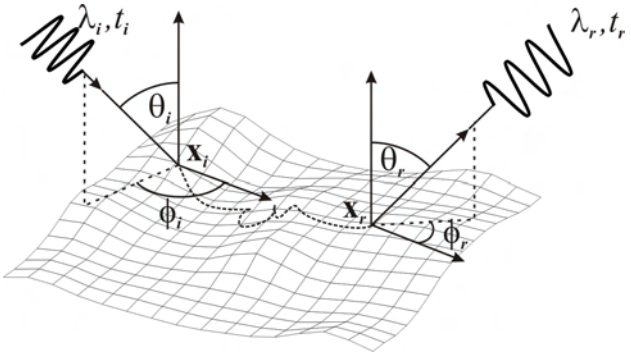



**Fig. 2.1: Light-matter interaction.**

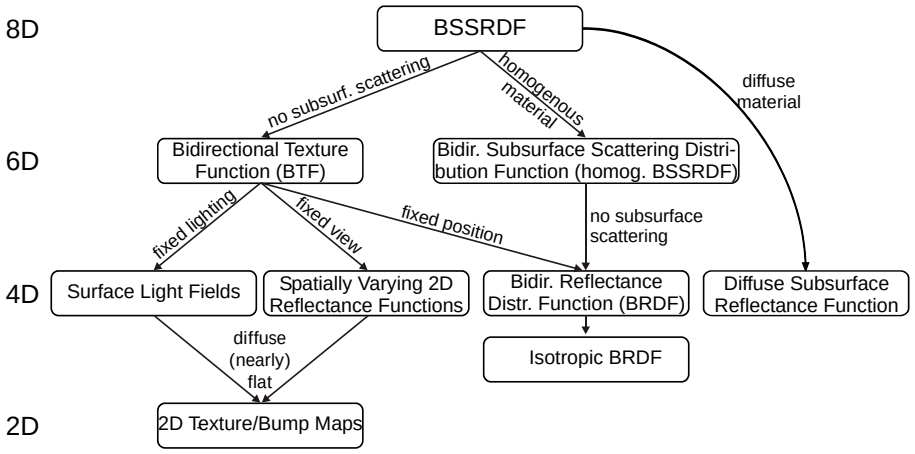



**Fig. 2.2: Material types.**

limited variant of wave optics since it neglects effects like interference, polarization, and diffraction. Despite this significant limitation, the function describing general light-matter interaction is still twelve dimensional and thus much too complex to be measured or modeled in a practical way. Therefore, typically further assumptions about the objects and their materials are made. First, they are assumed to remain unchanged over time and light transport is assumed to occur instantly, which removes dependencies on time parameters. Second, fluorescence effects are neglected. Third, interaction is modeled for a discrete set of wavelengths only (typical choices include the wavelengths of base colors of color models like RGB or CIE XYZ). The latter assumptions eliminate the dependencies on wavelengths. While this is clearly unacceptable for predictive rendering, which aims at reproducing spectral distributions of light, it turned out to be a reasonable choice for the current state of technology.

The resulting 8D function is called the *bidirectional scattering-surface reflection distribution function* (BSSRDF) [386]. It describes light transport between pairs of points on a surface for any incoming and outgoing direction. Due to its generality, typically only lower dimensional specializations of this 8D function are used in practice. Figure 2.2 gives an overview of derived functions describing the interaction of light and matter. Existing methods for modeling these functions are discussed in Section 2.1, omitting reflectance functions like (surface) light fields [169, 305, 363] or spatially varying reflectance fields [92, 338], since these cannot be considered *material representations*.

Unfortunately, even if reflectance properties of materials are modeled accurately this does typically not suffice for reproduction of realistic scenes. The problem stems from the spatial variation of most materials. While accurate modeling of reflectance properties for a material sample of limited size is well possible, objects in modeled scenes have arbitrary surface area, thus requiring arbitrarily large material samples. Typical mate-

rial models are not capable of capturing arbitrarily large extents, therefore additional techniques enlarging material samples are required. These texture synthesis methods – which aim at reproducing the reflectance properties including structural elements of material – are described in the second part of this chapter. Section 2.2 gives an overview of existing methods, comparing their strengths and weaknesses, and showing how they can be combined with high fidelity material representations. Additionally, this section gives a very brief overview of techniques for applying materials to surfaces, which is commonly referred to as *texturing*.

# 2.1 Material Models

## 2.1.1 Material Definition and Categorization of Material Models

In the above text, material was defined as a subsummation of geometric detail too small to be modeled explicitly. Although corresponding to the definition most commonly used in computer graphics, this description is very vague since it gives no indication on the size of detail covered by material. Therefore, this section first gives a more precise definition of material, especially elaborating on the differences between material and what is commonly referred to as *geometry*.

An intuitive distinction between material and geometry, which also corresponds quite closely to the most common definitions in dictionaries is the following: Material refers to the substance an object is made of while geometry determines an object's shape. Unfortunately, this definition does not lead to an absolute distinction since materials are often composed of materials themselves, making the term *material* application or scale dependent. As an example consider a sweat-shirt which has a specific shape (i.e., geometry) and is made of patches of cloth. The material cloth itself is made of is yarn which is knitted or weaved into panels (i.e., geometry). Yarn consists of cotton strands shaped as threads, etc.

A second meaning of the term *material* refers to a substance with characteristic properties such that naming or categorizing it becomes possible. For computer graphics, this implies that materials are surface or volume elements with (locally) homogeneous visual appearance properties, i.e., appearance properties that are distributed according to a characteristic, material-dependent distribution.

Predictive rendering aims at creating physically correct images by correctly simulating the interaction of light and matter. In the most precise setting, this requires simulation of the interaction of light waves or photons with the atoms and molecules the objects consist of. Since current rendering algorithms limit themselves to geometric optics, and since the laws of geometric optics are valid for sizes much larger than the wavelength only, the exact influence of nano-structures (i.e., size smaller than some nm) cannot be modeled accurately. Therefore, explicit modeling of geometry makes sense for larger structures only. The influence of nano-structures needs to be modeled by some material model, typically a function like the BSSRDF (cf. Figure 2.2).

Obviously, explicit geometric modeling of such nano-scale structures is inefficient for relatively large objects like cars or shirts due to the huge amount of geometric information. Therefore, typically the influence of micro-scale structures is included in the material model, yielding a micro-scale material model. As long as only structures invisible to the observer are included, this results in an almost perfectly accurate model.

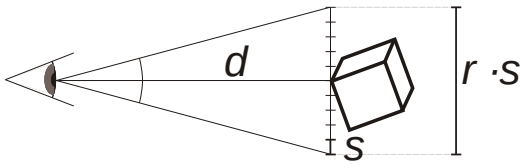



**Fig. 2.3: Minimum size of a structure visible on a display.**

For computer graphics, which displays images on a screen, this implies that structures need to be smaller than one pixel. Assuming perspective projection, the size

$$s = \frac{2d}{r} \cdot \tan \frac{\alpha}{2}.$$

of such a structure depends on the display resolution $r$ in each dimension, the distance $d$ between the observer and the structure, and the observer's field-of-view angle $\alpha$ (cf. Figure 2.3). As a realistic example, on a display with $2000 \times 2000$ pixels, viewing distance $d = 1$ m, and $\alpha = 30°$ field-of-view angle, structures smaller than approximately 0.54 mm remain invisible (note: in the worst case this corresponds to surface structures with a projected area of 0.29 mm$^2$).

While structures need to be small enough to be subsumed in a micro-scale material model, they also need to be typical for the material. Consider the case of a single spike placed on a perfectly planar infinite surface. The shadow cast by the spike leads to regions that look completely different than other regions, creating an inhomogeneous appearance, which contradicts one fundamental property of materials. Instead, if many spikes are placed on the plane, their shadows lead to a homogeneous appearance, making the inclusion of the spikes in a micro-scale model suitable.

For many materials like leather, knitted wool or gravel, modeling an object or scene by separating geometry and a micro-scale material model results in a huge geometric complexity. As an example, consider a sweat-shirt with a surface area of about 1m$^2$. If inspected with the above settings $d = 1$m, $r = 2000$, and $\alpha = 30°$, surface features like weaving or knitting patterns, which are larger than 0.54mm, remain visible and therefore require explicit modeling. Since geometric surface patches may cover at most 0.29mm$^2$, this requires a total of about 3.4M triangles. To reduce this geometric complexity, meso-scale material models were introduced. In addition to micro-scale geometry, they also capture visible geometry. The amount of geometry included depends on the desired tradeoff between material and geometry complexity. It ranges from inclusion of small features covering few pixels up to inclusion of all geometry[1]. Typically, the size of included features is less than ten pixels.

[1]Light fields [169, 305] model an object as a reflectance function (i.e., a material) of some surface enclosing the original object.

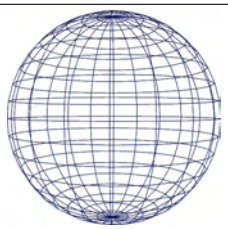
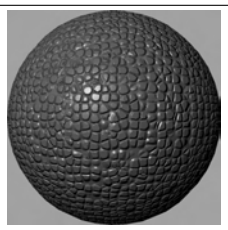
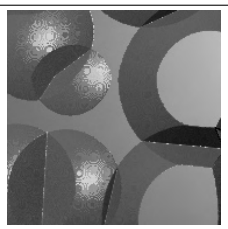
| | scale | structure size | model classes | |
|---|---|---|---|---|
|  | macro | $>$ 10 pixels | geometry models (triangles, parametric surfaces, etc.) | multi-scale models |
|  | meso | 1–10 pixels | textures, BTF, bump-mapping, displacement mapping | |
|  | micro | $<$ 1 pixel | BRDF, homog. BSSRDF | |

**Fig. 2.4: Categorization of material model classes.**

To optimize performance for real-time predictive rendering material models should be chosen such that an optimal tradeoff between material and geometry complexity is made. As pointed out above, this choice is not absolute but depends on the scale of observation: If objects are very close and display resolutions are high, the size of structures accurately represented by micro- and meso-scale material models is very small. For very distant viewing or low display resolutions, micro- and meso-scale models can include structures with much larger absolute sizes. Thus, suitable methods for transiting between different material models are required, which are typically part of multi-scale material models.

Table 2.4 summarizes the different material models, also introducing the distinction between micro- and meso-material models and geometry (macro-scale). Additionally, it names some of the prominent classes of models falling into the respective category.

In the following, existing members of these model classes are presented. The overview starts with micro-scale models, since they were developed first and since they provide a basis for the following meso-scale models (including the own contribution to meso-scale models described in detail in Chapter 3). The third section briefly covers multi-scale models since they are indispensable for smooth transitions between models of different scales.

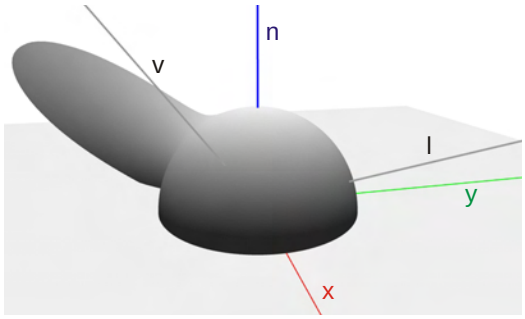



**Fig. 2.5: Geometry of BRDF models. A plot of a Phong BRDF for a fixed light direction is shown.**

### 2.1.2 Micro-Scale Material Models

As shown in Figure 2.2, until today two prominent micro-scale material models were derived from the fairly general model for light-matter interaction known as BSSRDF: The homogeneous BSSRDF, which considers subsurface scattering but assumes absence of spatial variations due to meso-scale structures or spatially varying micro-structures, and the *bidirectional reflectance distribution function* (BRDF) [386]. The BRDF additionally assumes that subsurface-scattering can be neglected. Due to their greater simplicity, BRDF models were proposed much earlier and are used more widely. Therefore, in the following first different BRDF models are reviewed.

#### Physically-Based BRDF Models

BRDFs are four-dimensional reflectance functions describing the ratio of exitant radiance into a specific direction $\mathbf{v} = (\theta_r, \phi_r)$ to the incident irradiance from a specific direction $\mathbf{l} = (\theta_i, \phi_i)$, both directions being specified w.r.t. a local coordinate system. They are typically defined on the upper hemisphere and their unit is $\frac{1}{sr}$. Due to physical requirements they are energy conserving and reciprocal [103]. BRDFs solely depending on polar angles $\theta_i, \theta_o$ and the difference $\phi_i - \phi_o$ of azimuth angles are called isotropic, while general BRDFs are called anisotropic. Figure 2.5 gives an overview of the parameters influencing BRDFs.

Accurate BRDF representation need to take into account the underlying micro-scale surface structure and the laws from physics describing light-matter interaction. Typically these are the laws of geometric optics, the Fresnel equations (which determine the amounts of reflected, transmitted and absorbed light), and dispersion curves [102]. Additionally, other laws from wave optics might need to be considered if the size of micro-scale structures is in the range of the wavelength of light [211]. While BRDFs for perfectly mirroring materials or Lambertian diffusers are easy to specify, BRDFs of existing materials are typically very complex. Therefore, very different approaches for

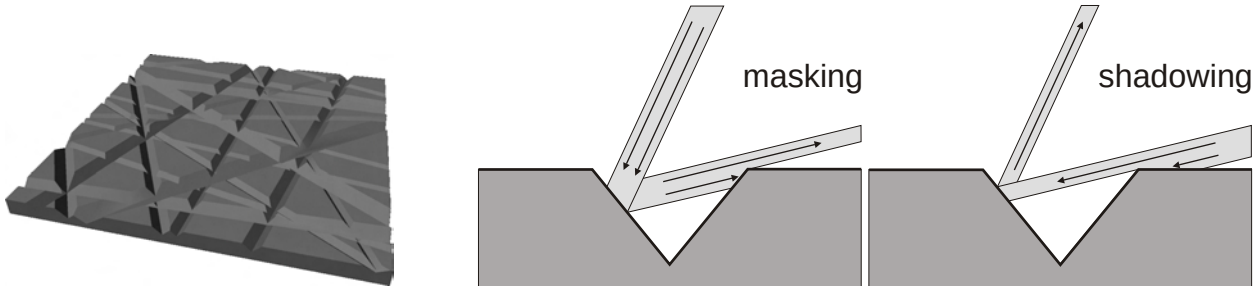



**Fig. 2.6: Microfacet model. Left: V-shaped grooves, right: shadowing and masking.**

modeling BRDFs have been developed.

A first group of such models is based on physics principles and assumptions about the small-scale structures interacting with light. The earliest model was introduced by Torrance and Sparrow in 1967 already [521]. It models the small-scale structures as a number of tiny V-shaped grooves (cf. Figure 2.6). Each groove consists of two microfacets with corresponding slopes, with reflectance properties equal to perfect mirrors. The amount of radiance reflected directly into a specific direction therefore depends on the orientation of microfacets, which is specified as a normal distribution function (here: Gaussian), and the Fresnel term. Due to masking and shadowing from rough surfaces, not all light reflected into some direction is observed by a viewer, which is modeled by an additional *geometry term*. The Torrance-Sparrow model is very successful at modeling BRDFs of a large class of materials but unfortunately modeling or fitting of respective parameters is difficult. Additionally, evaluating the model requires significant time. It was successfully applied to computer graphics by Blinn [35] and Cook and Torrance [78] who included an additional diffuse term to account for light being reflected multiple times. Cook and Torrance additionally assumed the normal distribution function to be a sum of individual functions, which allowed them to model color variations.

The model was later extended in several works. Poulin and Fournier [420] assumed parallel, cylindrical grooves instead of V-shaped ones. Shirley et al. [478] introduced a more accurate model for the matte term of polished surfaces, leading to highly specular behavior for grazing angles and almost diffuse reflections when seen head-on. Sun [504] introduced a roughness-dependent geometry term. He et al. [211] and Nayar et al. [375] extended the model to consider wave optics effects (i.e., polarization) by rederiving it from the Kirchhoff equations. The resulting model is more accurate but on the other hand much more complex and has thus hardly been used in computer graphics applications so far. Another model including wave optics effects was introduced by Stam [497]. It allows for efficient simulation of anisotropic diffraction effects from metallic surfaces due to an analytic solution of the Kichhoff integral for these cases.

Other researchers simplified the model for practical reasons. Schlick [453] approximated the Fresnel term, the geometry term and the normal distribution function by rational fractions which can be evaluated very efficiently. The resulting model shows a good trade-off between accuracy and efficiency. Ashikhmin et al. [12] employed a simpler geometry term directly derived from a normal distribution function. This allows them to specify the geometry term in a very simple way and leads to faster evaluations while hardly impacting rendering quality for a large number of materials. Kelemen and Szirmay-Kalos [263] proposed a simplified version based on an approximated geometry term which is better suited for importance sampling. Due to such simplifications, the microfacet model is still widely used in computer graphics applications today.

A very interesting instantiation of the microfacet model was used by Oren and Nayar [399]: Instead of modeling directionally dependent reflections they aimed at determining a more accurate model for diffuse reflections. They assumed the microfacets to behave as perfect diffusers obeying Lambert's law instead of acting as perfect mirrors. The resulting model leads to significantly brighter regions where the light is almost parallel to the surface, which is in accordance with real observations.

### Empiric BRDF Models

Derivation of BRDF models from physics principles has the significant advantage of resulting in (approximately) correct models. Unfortunately, the models usually feature several parameters which are difficult to determine (i.e., the distribution of normals of microfacets) which makes user-guided design of BRDFs or fitting to existing data very difficult. To improve on these problems, less complex empiric models were developed. They allow for efficient manual modeling and fast evaluation since they depend on rather intuitive parameters. Empiric models are not derived from physics considerations but solely attempt to replicate observed reflectance properties.

In 1975, Phong proposed a model based on splitting the reflected intensity into two different parts: diffuse[2] and specular (also: directed-diffuse) intensity. While the diffuse component is computed from Lambert's law, the directed part is modeled by a cosine lobe, with the lobe exponent related to the specularity of the material. The Phong BRDF can efficiently be evaluated but is neither energy-conserving nor reciprocal nor does it allow for accurate modeling of real-world BRDFs. Blinn [35] introduced a change to the model which establishes a link between the Phong BRDF and the microfacet BRDF models. In the changed model the lobe can be considered a normal distribution function, thus leading to more accurate results, but the model still lacks accuracy due to missing

[2] As opposed to the notion of diffuse reflection in microfacet models, the diffuse term in the Phong model does not correspond to light multiply reflected by the material surface.

Fresnel and geometry terms. A further improvement was suggested by Lewis [307], who devised changes such that the BRDF becomes energy conserving and reciprocal.

Another set of changes was introduced to make the Phong BRDF anisotropic. Ward [565] proposed anisotropic lobes modeling surfaces with a Gaussian normal distribution. Fitting his model to measured BRDF data lead to very good results. In a following application, Ashikhmin and Shirley [11] mixed ideas from the Ward and Schlick models: They used anisotropic lobes similar to Ward, Schlick's Fresnel term approximation, and included view-angle-dependent influences of the diffuse part. They also showed that their model can efficiently be used in Monte-Carlo raytracing applications since it allows for efficient importance sampling. Another generalization of the original model was proposed by Lafortune et al. [290]. Instead of using simple cosine lobes as Lewis [307], their model allows for anisotropic lobes while retaining energy conservation and reciprocity. Their model has been fit successfully to measured data using non-linear optimization methods [89, 348].

### Data-Driven BRDF Models

Whereas empiric models usually contain few parameters only and are thus highly suitable for BRDF modeling, their accuracy limits their use for predictive rendering [384]. Since modeling of physically-based BRDF models is very difficult, researchers suggested measurement of BRDFs by densely sampling BRDF values for varying light and view directions, resulting in tabulated BRDFs. While most publications deriving such data focus on setups measuring BRDFs of real materials [302] it is also possible to compute BRDFs from modeled micro-scale surface geometry using appropriate rendering techniques [52, 166]. A key problem of tabulated BRDFs is that they do not represent continuous functions any more, requiring interpolation of sampled BRDF values. A second problem are the memory requirements for storing dense samples of 4D functions. To resolve these problems, existing BRDF models can be fit to measured data or tabulated BRDFs can be transformed into more suitable bases like spherical harmonics [52, 577], spherical wavelets [460], wavelets [291], or Zernike polynomials [275].

Another popular representation for tabulated BRDFs is based on separable BRDFs, i.e., BRDFs that can be modeled as products of functions $g(\mathbf{l})$, $h(\mathbf{v})$ depending on incident or exitant direction only, since they enable very efficient rendering algorithms (e.g., separable BRDFs can efficiently be integrated into radiosity solutions [379, 153] which is not the case for general 4D BRDFs). A first such model for the BRDF of the moon was proposed by Minnaert [365] in 1941 already. Later this approach was introduced to computer graphics by Neumann and Neumann [379] who specified a separable BRDF for diffuse materials covered with a layer of lacquer. Unfortunately, both approaches are

very limited in their expressiveness since their peak reflection directions always point into the direction of the surface normal. This problem was removed by Fournier [153] who suggested to model an arbitrary BRDF by a sum of products of 2D functions

$$BRDF(\mathbf{l}, \mathbf{v}) \approx \sum_j g_j(\mathbf{l}) h_j(\mathbf{v})$$

which are computed using singular value decomposition (SVD). To improve the separability of BRDFs, Rusinkiewicz [444] suggested to reparameterize the BRDFs w.r.t. the basis found by Gram-Schmidt orthonormalization of $(\mathbf{h}, -\mathbf{n}, \mathbf{n} \times \mathbf{h})$ ($\mathbf{h}$ being the halfway vector from the Blinn-Phong model, and $\mathbf{n}$ the surface normal). Unfortunately, this reparameterization becomes instable whenever $\mathbf{n} \approx \mathbf{h}$. Therefore, Kautz and McCool [258] suggested a basis resulting from Gram-Schmidt orthonormalization of $(\mathbf{h}, \mathbf{t}, \mathbf{s})$ ($\mathbf{t}$ and $\mathbf{s}$ being the original tangent and bitangent). The new basis allows reparameterization values to be interpolated over triangles which enables efficient rendering per fragment using commodity graphics hardware. A further improvement was introduced by McCool et al. [349] who represent the BRDF as

$$BRDF(\mathbf{l}, \mathbf{v}) \approx e^{\Pi_j g_j(\pi_j(\mathbf{d_i}, \mathbf{d_r}))},$$

with projection functions $\pi_j$. The approach leads to optimization of relative errors (i.e., log scale errors) which corresponds to the approximately logarithmic luminance perception of the human visual system. An additional advantage is that separate projection functions can be applied per component, allowing extraction of various BRDF features. A combination of the latter two techniques was presented by Suykens et al. [507] which results in functions that are better suited for rendering on integer graphics hardware.

Having suitable compression methods for tabulated BRDFs at hand, use of such data became increasingly popular. Unfortunately, modeling of tabulated BRDFs is a very difficult problem due to the large number of samples to be specified. To solve this problem, Matusik et al. [345] developed a method for meaningful inter- and extrapolation of available data. They acquired a large number of isotropic BRDFs and determined manifolds (i.e., subspaces with lower dimension) in the space of measurement values that represent the measured data points well. Navigation on these manifolds allows for reasonable mixtures of BRDFs. The authors additionally learned navigation directions like 'increased diffuseness' or 'metallic look' from user characterizations of measured BRDFs, allowing for more intuitive navigation in the space of BRDFs. A comparable approach based on intensive user studies is described by Pellacini et al. [409].

Recently, two new approaches for modeling complex BRDFs were proposed. Lawrence et al. [295] semi-automatically factorize a spatially varying BRDF into more easily ed-

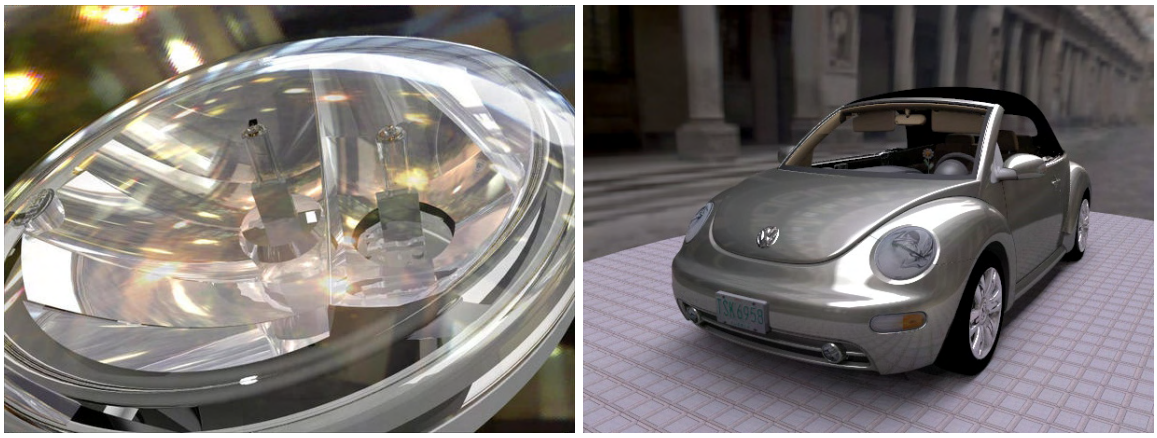


**Fig. 2.7: Rendering with BRDFs. Left: car headlight, right: car body with measured car paint. Images from [27, 179]**

itable components like textures and 1D or 2D BRDF components which can be modeled by simple curves. Ngan et al. [385] suggested a BRDF modeling system based on selecting images showing a simple scene with objects covered with slightly varied versions of the currently selected BRDF, allowing directed searches for desired BRDFs. Although the BRDF parameters for rendering navigable images are chosen according to an perceptually motivated metric, the system is not capable of handling BRDFs as complex as those described by tabulated BRDFs. The same holds for the approach of Lawrence et al.

In summary, tabulated BRDFs have been the most preferential BRDF representation for high-quality rendering, leading to extremely realistic rendered images (cf. Figure 2.7). Methods for modeling such data have been developed already but are currently not sufficient for all purposes. Due to the large importance of this kind of data, it seems very likely that better modeling techniques will be developed in the future.

### Homogeneous BSSRDF

Many materials can be represented by BRDFs very accurately and efficiently. Unfortunately, the BRDF is limited to reflection effects occurring at the object surface. This approach fails to model materials like most fluids, glass, or marble since their appearance is largely governed by subsurface scattering.

Hanrahan and Krueger [200] were the first to simulate subsurface light transport and participating media using accurate Monte-Carlo algorithms, leading to very accurate results that require extensive computation time. A first practical model of a homogeneous BSSRDFs was later introduced by Jensen et al. [241]. They efficiently approximated the single scattering term by applying a dipole approximation of material to the light diffu-

sion equation. Good results were achieved for materials like marble, skin, or milk, and later publications even rendered such results at interactive rates [240, 301, 355, 201].

A special class of materials showing significant subsurface scattering effects and efficiently described as homogeneous BSSRDFs are multi-layer materials (e.g., lacquered surfaces). They are special since they assume the lowest layer to be opaque, yielding a mixture of surface and volumetric reflectance behavior. Due to the slender thickness of the layers, often wave-optics effects like wavelength and direction dependent reflection, refraction, interference, and absorption need to be considered for accurate simulation of the layered material. Practical approaches [200, 220, 230] consider layered materials as a single system, which removes the necessity to handle individual layers, but require spectral rendering to achieve realistic effects. A notable extension is the approach of Granier and Heidrich [173] which creates approximate yet reasonable effects within RGB rendering systems. A similar yet more sophisticated multi-layer reflection model is described by Donner and Jensen [113]. It extends the dipole approximation for translucent materials to thin layers to enable successful simulation of materials like human skin, including a large number of effects like reflectance and transmittance of skin layers, and influences of blood vessels.

### Discussion

Micro-scale models are fairly accurate for describing micro-scale interactions of light and matter. BRDFs give very good results for materials without subsurface scattering. Currently, the best possible quality is achieved using tabulated BRDFs, since they accurately capture reflectance properties without requiring explicit, parametric models. Thus, they are especially suitable for predictive rendering. Unfortunately, they also feature a significant problem: Practical creation of tabulated data requires measurement of existing data due to the large number of tabulated values. Existing editing techniques for tabulated BRDFs are not reliable and accurate enough to be used in real-world applications. Therefore, users still tend to employ empiric BRDF models since their parameters can be edited in a relatively intuitive way. Additionally, existing graphics hardware allows these changes to be taken into account in real-time even if complex models are rendered.

For materials showing significant subsurface reflectance behavior, a number of accurate models have been introduced, many of which can be evaluated very efficiently. While the accuracy of models for multi-layered materials is typically quite high, true, practical homogeneous BSSRDF models rely on the dipole approximation of single scattering. Obviously, this approximation and the avoidance of multiple scattering terms limit the correctness of rendered images.

**Fig. 2.8: Slices of a ceiling panel BTF. Left: application of BTF to sphere, right: several views of the BTF for varying view and light directions.**

### 2.1.3 Meso-Scale Material Models

In contrast to micro-scale material models, meso-scale material models include structures visible to the observer. As a consequence, they are capable of capturing spatial material variations, either due to spatial material changes (e.g., changes of material color or material type) or structures in the material. The earliest meso-scale models simply combined BRDFs with color textures to simulate spatially varying material. The varying colors stored in textures can replace or scale the average color of BRDFs. Unfortunately, such an approach omits the effects due to mesoscopic structures.

#### Models Capturing Spatially Varying Reflectance

A more complex material representation capturing meso-scale structures and spatial materials changes is the *bidirectional texture function* (BTF), which was first introduced by Dana et al. [87]. As Figure 2.2 shows, it is also a simplification of the BSSRDF. The relationship between the BSSRDF of an object with surface $S$ and the BTF can be expressed as follows:

$$\mathrm{BTF}(\mathbf{x}_i, \mathbf{l}, \mathbf{v}) = \int_S \mathrm{BSSRDF}(\mathbf{x}_i, \mathbf{l}, \mathbf{x}_r, \mathbf{v}) d\mathbf{x}_r.$$

Thus the BTF contains all effects resulting from mesoscopic structures on the captured material sample (i.e., interreflection, self-shadowing, self-occlusion) and even subsurface scattering effects, although subsurface scattering is not modeled explicitly. Like BRDFs, BTFs are typically specified by tabulating values for surface locations, incident and exitant directions. The large number of necessary values can either be acquired from real materials using suitable measurement setups (for an overview see [371]) or by ray tracing surfaces with modeled surface geometry [108, 89].

Due to the large amount of tabulated samples (typically several GBs [371]) the necessity for data compression is even more pressing than for tabulated BRDFs. Consequently, a large number of BTF compression methods has been developed, which

can roughly be categorized into two groups. The first group applies BRDF modeling techniques to approximate the BTF values per surface location. While McAllister et al. [348] employ Lafortune lobes per texel, Daubert et al. [89] fit an additional view-dependent scaling factor to account for mesoscopic occlusion effects. The necessity to respect the fact that per-pixel values do not represent BRDFs but rather *apparent BRDFs*[3] (ABRDFs) [584] is highlighted in the method presented in the next chapter, which is based on *reflectance fields* (also called *spatially varying 2D reflectance functions*, cf. Figure 2.2). A similar technique based on the same principle but less suitable for real-time rendering on graphics hardware was published by Filip and Haindl [145].

The second group of methods originates from data-driven compression of tabulated BRDFs. Suykens et al. [507] compressed BTFs by factorizing per-texel ABRDFs and additionally clustered factorized values to exploit the coherence of ABRDFs. Unfortunately, this approach either leads to visible clustering artifacts or to insufficiently reduced storage. A more promising approach was taken by Liu et al. [319] and Koudelka et al. [278] who factorized the full BTF into spatially varying functions $g_i(\mathbf{x})$ and functions $h_i(\mathbf{v}, \mathbf{l})$ depending on incident and exitant directions. While this leads to very good compression rates, the approach has two problems. First, factorization of the full BTF data requires concurrent handling of huge amounts of data. Thus, often costly out-of-core factorization methods are required which implement some kind of external memory management. Second, a large number of functions needs to be retained to achieve high-quality reconstruction, which hampers use for real-time rendering. Therefore, other approaches factorize subsets of the full data. Sattler et al. [447] applied per-view factorization, i.e., they applied SVD to BTF slices with fixed view directions. While this approach leads to a significantly reduced number of retained components for high-quality reconstruction, the overall compression ratio of this method is much worse since correlations between different view directions cannot be exploited. Another approach handling subsets of the full data was published by Müller et al. [369]. It performs factorization of clusters of ABRDFs where the clusters are chosen such that the reconstruction error from the factorized representation becomes minimal. Compared to per-view factorization even fewer components need to be retained and the overall compression ratio is similar to the full matrix factorization approach. The most recent approaches for compression of BTFs are based on tensor decomposition [532, 553] which allows for selective compression in selected dimensions (e.g., spatial dimension, polar light angle dimension, etc.). While this helps to achieve less perceptual error, the reconstruction costs for this approach are significantly higher than from other factorized representations, making the approach less suitable for real-time applications.

[3] ABRDFs are 4D functions of view and light direction like BRDFs but obey neither reciprocity nor energy conservation.

Comparing the two groups of compression methods is difficult. While methods based on data-driven compression typically lead to better tradeoffs between compression ratio and reconstruction quality, methods based on BRDF modeling achieve higher rendering speed since they avoid interpolation of sampled values. Therefore, both approaches are employed in practice today. A more detailed comparison of existing methods including a thorough description can be found in the survey of Müller et al. [371].

An important aspect for spatially varying materials is the support of multiresolution representations. While only some of the above compression approaches provide explicit support for MIP mapping of BTFs, a specialized technique based on a Laplace pyramid was presented by Ma et al. [330]. Unfortunately, this approach requires increased computational effort for evaluating detailed BTF views.

### Models Capturing Height Variations

Besides the techniques modeling spatially varying reflectance behavior due to local changes in material, other techniques were developed which specifically aim at modeling the influence of mesoscopic structures independently of the material's BRDF. Already in 1978 Blinn showed that the impression of a rough surface can be achieved by varying per-fragment normals [36] since illumination models consider normals for shading computations only. His ideas lead to different bump- and normal-mapping techniques which can efficiently be integrated into real-time rendering approaches on existing programmable GPUs. Bump- or normal maps can either be specified by the user, can be derived from actual geometry, or can be acquired from photographs using shape-from-shading techniques [442].

An important problem for bump-mapping is the support of multiple resolutions. Standard approaches like MIP-mapping fail for two reasons: First, normals have non-linear influence on shading computations, prohibiting linear prefiltering. Second, averaged normals introduce false or missing highlights since they correspond to normals derived from surface smoothing operations. Obviously, the shading of a fragment should not correspond to a smoothed surface but to the original surface under the fragment's footprint [451, 392]. To overcome these problems, researchers used normal distribution functions (NDF) instead of averaged normals. Fournier [152] modeled the NDF by a sum of a varying number of Phong lobes. Due to the non-linear influence of lobe parameters and the varying number of lobes, this model unfortunately prohibits bi- or trilinear interpolation required for spatial antialiasing. A better model was proposed by Olano and North [396] who modeled the per texel NDFs by 3D Gaussians. Gaussian parameters can be interpolated linearly but evaluation requires an expensive normalization step – which can fortunately be precomputed – and leads to more complicated

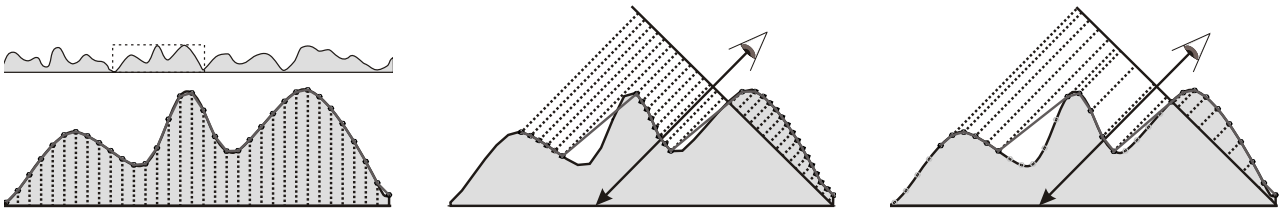



**Fig. 2.9: Relief mapping of a part of a surface with height variation (left top). The depth values in the texture approximate a surface as seen from the front. When seen from a different angle, they are resampled into a new texture – which is applied to the rendered surface – using forward or backward warping.**

shading equations. A similar approach was followed by Schilling [451] who stored normal covariance matrices in a roughness map. The approach leads to a slightly modified Blinn-Phong shading model which requires the angle governing specular reflection to be increased or decreased based on the respective covariance matrix. A simpler approach by Toksvig [392] simply MIP-maps 3D normals and estimates normal variation from the lengths of the unnormalized interpolated normals. Depending on the amount of variation scale factors and changed exponents for the Phong BRDF model are selected. A very recent contribution to the problem was presented by Tan et al. [513]. They model NDFs by Gaussian mixture models, which allows for efficient rendering with an extended Cook-Torrance model.

Compared to BTFs, bump or normal maps feature the advantage that they require much less storage and that respective shading algorithms are more efficient to implement. In addition, inclusion of sharp surface features using vectorial texture maps [433, 514] is much simpler than for BTF rendering. As the main disadvantage, they fail to capture spatially varying materials, i.e., they need to be combined with spatially varying BRDFs to achieve comparable effects as BTF rendering methods. But even such combinations omit the influence of mesoscopic structures: Rendering of local self-shadowing requires techniques like horizon mapping [346], which stores for every texel the angles at which surrounding geometry is occluding incoming light. While this helps to add shadowing, it fails to model local self-occlusion and subsurface scattering.

To improve upon the self-occlusion problem, Oliveira proposed relief-mapping [398], a technique that warps the surface texture to match the current view before mapping it onto the surface (cf. Figure 2.9). To perform the warp, the texture stores additional depth values. This approach leads to convincing parallax effects but users need to take care to avoid disocclusion artifacts. An additional problem is that the required forward warping techniques [350] can not efficiently be evaluated on graphics hardware. These problems are avoided by the parallax mapping technique proposed by Kaneko et al.

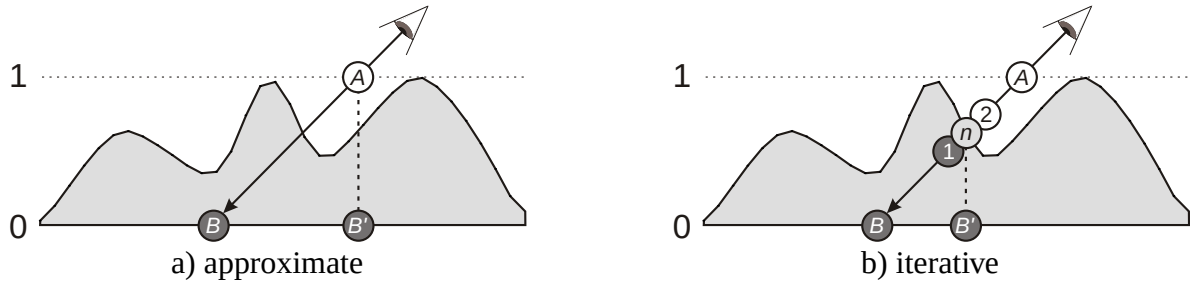



**Fig. 2.10: Parallax mapping. The texture coordinate $B'$ used instead of $B$ is either computed from the intersection $A$ of the upper limit surface and the view ray, or by an interval nesting scheme which iteratively checks intervals $[A, B]$, $[A, 1]$, $[1, 2]$, ... until determining the intersection point after $n$ steps.**

[252] (cf. Figure 2.10a). The approach changes the texture coordinates of fragments depending on the expected parallax. Unfortunately, parallax can only be estimated very roughly, leading to very approximate solutions. To resolve this problem, Policarpo et al. [417] recently proposed a GPU implementation which determines an exact value by computing intersections of view rays and the height-field surface by a binary search algorithm (cf. Figure 2.10b). The algorithm enables real-time frame rates since the height variations of the material are small, leading to small search spaces. The approach was recently extended to non-height field geometry [416] by employing several layers but still leads to problems for view rays almost parallel to the surface since search spaces become infinitely large for this case.

An approach handling height variations in the most general case is *displacement mapping* (DM), which was first proposed by Cook [76]. Instead of storing varying normals in textures as for normal mapping, texture values specify surface displacements along the normal. Unfortunately, while DM is very powerful since it leads to efficient decoupling of geometric modeling at various scales, so far no general real-time evaluation methods other than those based on fine triangulations of the base surface exist. Several researchers therefore concentrated on specialized, real-time implementations of DM. Schein et al. [450] proposed an implementation that achieves real-time frame rates for repetitive displacement patterns like spikes or thorns. Wang et al. [556] handled inward displacements with limited height variation using *View-Dependent Displacement Maps* (VDDMs). For every surface point, each view direction and each curvature value in the direction of viewing, the VDDMs store the distance from the surface point to the intersection point of the view ray and the displaced surface (cf. Figure 2.11). During rendering this information is used to change the surface texture coordinates in a view-dependent way (if the ray intersects the displaced surface at all), leading to accurate parallax effects and accurate silhouettes. VDDMs can be compressed using SVD factorization techniques and thus require acceptable amounts of memory. Unfortunately,

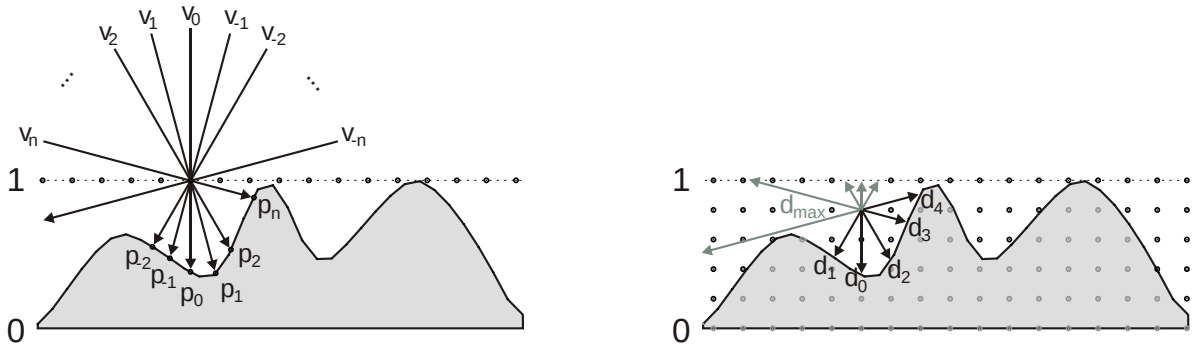



**Fig. 2.11: Real-time displacement mapping. Left: View-dependent displacement maps [556] sample distances to the displaced surface for points on the offset surface. Right: Generalized displacement maps [561] also sample at points in the volume if they are not contained within the material (gray points).**

displacements are restricted to height-field geometry, which turns out to be a bad assumption for many materials, and reconstruction quality strongly depends on a highly uniform surface parameterization. The restriction were relaxed by the *Generalized Displacement Mapping* technique of Wang et al. [561] which samples distances in the material volume instead of on the surface (cf. Figure 2.11). The approach allows for higher fidelity rendering on graphics hardware using finely triangulated meshes, leading to very realistic material renderings. Such high-quality displacement mapping techniques can additionally be combined with BTF rendering methods [554] to combine the strengths of both techniques (i.e., to extend BTF rendering with accurate silhouettes and non-planar shadows).

### Specialized Extensions

Similar in case to the BRDF micro-scale models, the meso-scale models reviewed so far are limited to surface reflectance phenomena. To extend the application area to inhomogeneous materials showing subsurface scattering effects, Chen et al. [63] introduced *Shell Texture Functions*. These model objects made of heterogeneous materials by a homogeneous interior and a heterogeneous outer shell. While light transport within the inner layer is handled with Jensen et al.'s dipole approximation for homogeneous materials [241], light transport in the outer shell is determined from precomputed data of measured materials. Shell Texture Functions lead to very realistic rendering of thick objects and can approximately be rendered in real-time [493]. For thin objects the method of Tong et al. [519] should be used, which is based on explicit acquisition of reflectance and transmittance properties of thin, inhomogeneous materials.

Of course many other specialized material models have been developed for all kinds

of materials (e.g., metal, skin, or fur). Several others try to model the influence of aging effects like weathering or rusting on the appearance of a material. An overview of such approaches can be found in the recent survey of Dorsey and Rushmeier [114] and the paper of Gu et al. [177].

### 2.1.4 Multiscale Models

The above definition of micro-, meso-, and macro-geometry relates the three scales to sizes of displayed projections. As a consequence, the absolute size of structures encoded in micro- and meso-scale materials, and macro-scale geometry depend on observer parameters. Thus, for efficient high-quality rendering, it is necessary to adjust the resolution of meso-scale material and macro-scale geometry (cf. Section 2.1.3 and Part III), and to switch between different representations for materials and objects contained in a rendered scene. As an example, Becker and Max [24] proposed the use of BRDF models for microscopic structures, bump mapping techniques as long as the missing silhouettes are not noticeable, displacement mapping for larger structures, and geometry for large-scale shape. Obviously, such an approach may minimize the required rendering time but the techniques can only be used in parallel if changes between the scales introduce no noticeable artifacts.

In a first approach targeting this problem, Cabral et al. [52] efficiently derived BRDFs from displacement-mapped surfaces using horizon mapping. This enables smooth changes between the two techniques including shadowing. Unfortunately, other effects like self-occlusions or self-interreflections cannot be modeled. More accurate BRDFs were derived by Westin et al. [577] using Monte-Carlo raytracing. In addition to the previous approach, they take into account transmission effects, allowing for limited subsurface scattering simulations. In addition to displacement maps and BRDFs Becker and Max [24] included bump mapping as an intermediate rendering algorithm. To compensate for missing self-occlusions in bump-mapping, they introduce *redistribution mapping*, a technique based on view-dependent bump normals. In addition, they derived thresholds for switching between rendering techniques based on viewing distances and surface structure sizes. Techniques for additionally considering self-interreflections were proposed by Heidrich et al. [214] and Daubert et al. [88], leading to quite accurate rendering results for surfaces with mesoscopic height variations.

Due to steadily increasing shading abilities of graphics hardware, multi-scale material models became less attractive in the last years. Nevertheless, due to the significant reductions in rendering time that can be achieved with these approaches, it seems very likely that their importance will be rediscovered in the near future.

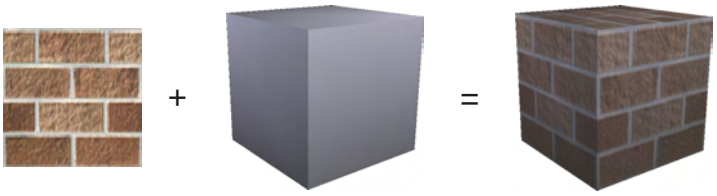


**Fig. 2.12: Texturing a surface.**

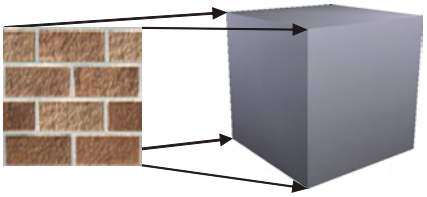


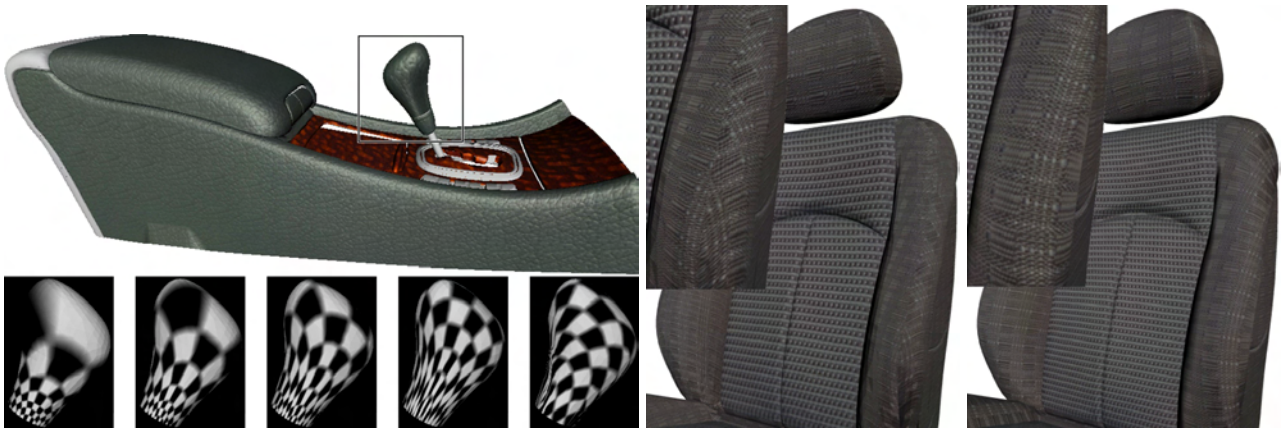
**Fig. 2.13: Surface parameterization.**

## 2.2 Material Synthesis

While accurate modeling of material properties is an essential step towards creation of physically correct images, it does not solve all practical problems related to materials. One problem left unsolved is application of materials to geometric surfaces, which is typically called *texturing* (cf. Figure 2.12). Texturing assigns a material coordinate $\mathbf{m} \in M$ (typically from a plane, i.e., $M \subset \mathbb{R}^2$) to each surface position $\mathbf{s} \in S$ (typically $S \subset \mathbb{R}^3$). Such a mapping is commonly derived by inverting surface parameterizations $\Phi : M \rightarrow S$ (cf. Figure 2.13), which are implicitly defined for parametric surfaces and which can be determined for other kinds of surfaces using parameterization algorithms [147]. In order to mimic the process of applying real materials to real surfaces, such parameterizations need to fulfill several requirements, two of them being consistency and continuity (cf. Figure 2.14). The first requirement enforces that materials are stretched or compressed almost equally when applied to a surface which mimics the physical behavior of materials. The second requirement demands that connected surfaces be textured by single, large patches of material, thus minimizing the number of required cuts and seams. Determining parameterizations fulfilling these requirements is a very complex task but fortunately quite successful approaches exist already [100, 147].

Almost all types of materials existing in reality show significant spatial variation. Variations are introduced both by spatially varying distributions of materials (e.g., textiles often consist of threads of different colors) and material structures (e.g., textiles show weaving or knitting patterns). As shown in the previous section, different approaches for modeling of spatially varying materials exist. The most accurate models are based on acquired reflectance properties from existing material samples. Due to the large amount of data to be captured and due to physical limitations of the size of acquisition setups, only samples of limited size can be represented. Unfortunately, this limitation conflicts with the continuity requirement of surface parameterizations, which demands availability of material samples with arbitrarily large extent.

To resolve this conflict methods for enlargement of material samples were developed. In the following, an overview over these methods – typically called *Texture Synthesis*



varying texture stretch undesired cuts continuous texture

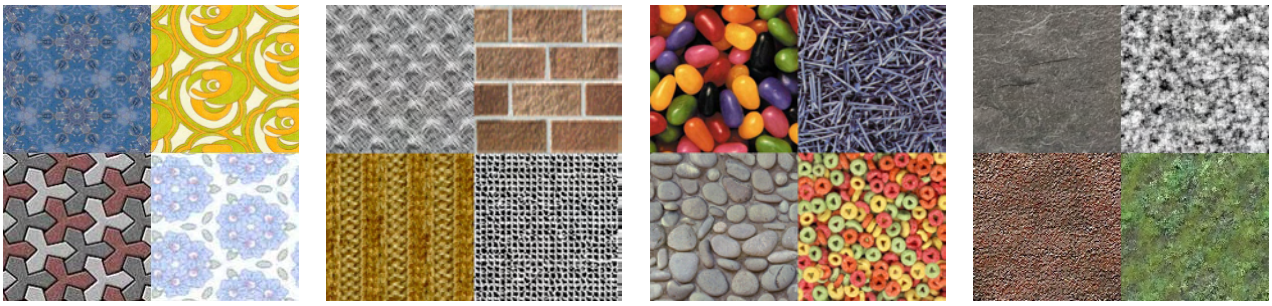
**Fig. 2.14: Realistic material application requires consistency (left: texture stretch on the gear shift lever) and continuity (right: two versions of a textured seat including zooms on regions of interest).**

methods – is given. The overview first covers methods for synthesis of diffuse textures, since they contain the principle ideas required for synthesis of arbitrary, spatially varying materials, and since they were developed first. Then, the overview briefly reviews texture tiling methods since they represent efficient ways for texturing large surfaces with minimal storage requirements. Next, methods for synthesizing materials directly on surfaces are covered since they represent an efficient alternative to combinations of texture synthesis and surface parameterizations, and additionally allow for efficient texturing of large areas like the tiling approaches. Finally, generalizations of these approaches to more complex materials (i.e., BTF material representations) are presented.

### 2.2.1 Texture Synthesis

The earliest methods for synthesis of materials focused on a quite restricted class of materials: diffuse textures. The goal of texture synthesis methods is generation of larger material samples with reflectance properties considered typical for the material being enlarged. To formalize this goal, textures are considered representative cutouts from materials of infinite extent which are composed of stochastically repeating elements. These two assumptions give rise to texture modeling by stationary, local stochastic processes. Typically, these processes are described by Markov Random Fields (MRFs) [81] or the closely related Gibbs Random Fields [162]. Texture synthesis algorithms therefore target generation of new textures which appear to be generated by the same stochastic process as the input texture.

Due to the large variety of existing textures (cf. Figure 2.15) very different texture



(a) regular (b) near-regular (c) structured (d) stochastic

**Fig. 2.15: Texture types. Regularity decreases from left to right.**

synthesis algorithms were developed, some of them specialized on specific types of textures, some being very general. Roughly, these approaches can be divided into two categories: approaches explicitly modeling the stochastic processes, and approaches omitting this explicit modeling step.

### Model-Based Texture Synthesis

Results from the first category were developed earlier, probably since they are closely related to approaches for texture analysis, segmentation and classification – well-established topics in the computer-vision community. Existing techniques try to learn a statistical model for the stochastic process that generated the texture and then synthesize new texture by adequately sampling this process. Like the BRDF material models, these texture models can be divided into parametric and non-parametric ones.

Approaches employing parametric models base on the same principle as procedural textures [410, 524, 583], which are frequently used to model solid textures like marble of wood: The texture value at any location is determined by evaluating a parametric formula, typically requiring a small number of parameters only, which allows for memory efficient storage of textures. While procedural textures rely on user specified parameters, texture synthesis approaches determine such parameters from an input texture which is supposed to be representative for the underlying stochastic process. Existing approaches vary in the mathematical model for this random process (typically variants of MRFs [310] or autoregressive models [26]) and in the way these models are represented. Most approaches assume Gaussian distributions of gray scale or color values and thus represent the processes by Gaussian mixtures [195]. Unfortunately, as Figure 2.15 shows, different types of textures show significantly different characteristics. Therefore, capturing all kinds of textures by a single parametric model seems to be impossible, especially since efficient estimation of required parameters is a very difficult task. Researchers often resolve this problem by factorizing color textures into several

channels that can be processed individually [495, 196] but this does not solve the problem completely.

To avoid the difficulty of determining adequate parameters, non-parametric texture models were developed. Heeger and Bergen [212] characterize textures by statistical measures of a wavelet representation. First, the texture is transformed into a *steerable pyramid* by applying several oriented filters to the input texture and building Laplacian pyramids of the filter responses. Then, histograms for all layers of the pyramids are computed. New textures are synthesized in a multiresolution fashion starting from a Laplacian pyramid initialized with random values. Iteratively, the histograms of the levels of this pyramid are matched with the histograms of the input texture until a close match is achieved. Relatively good results are possible for stochastic textures but the approach fails for structured textures since structures are difficult to reproduce using statistics of local features. Additionally, synthesis times are very long since dependencies between pyramid layers are not considered. This was resolved to some extent by the approach of Paget and Longstaff [402]. As a third problem, the quality of results largely depends on the choice of filters applied to the image and the type of statistics evaluated. Simoncelli and Portilla [485] and Fan and Xia [136] showed that the use of additional second order statistics leads to improved results. This behavior was predicted by Julesz [247] much earlier already. He claimed that $n$-th order statistics should lead to an accurate texture classification such that texture classes are indiscernible by human observers. Unfortunately, an exact number for $n$ could not be given.

All approaches relying on explicit modeling of the stochastic process generating the texture show significant success at synthesizing unstructured textures, since they are optimized for reproducing statistical distributions of colors. Unfortunately, due to the immanent difficulty of capturing structure by (local) statistical analysis, these methods have difficulties handling structured or even (near-)regular textures (cf. Figure 2.15).

### Sampling-Based Texture Synthesis

The second category of texture synthesis algorithms avoids computation of parameters characterizing the stochastic process that generates the input texture. Instead, respective algorithms intelligently sample from the input texture, i.e., they copy parts of the input texture into the synthesized result (cf. Figure 2.16). Methods from this category are usually discriminated by the parts they copy – either individual pixels or whole patches (i.e., image-regions).

Pixel-based texture synthesis algorithms were introduced by the work of De Bonet [91]. His method first creates a steerable pyramid from the input texture. Synthesis of new textures starts by copying the value from the lowest resolution into the respective

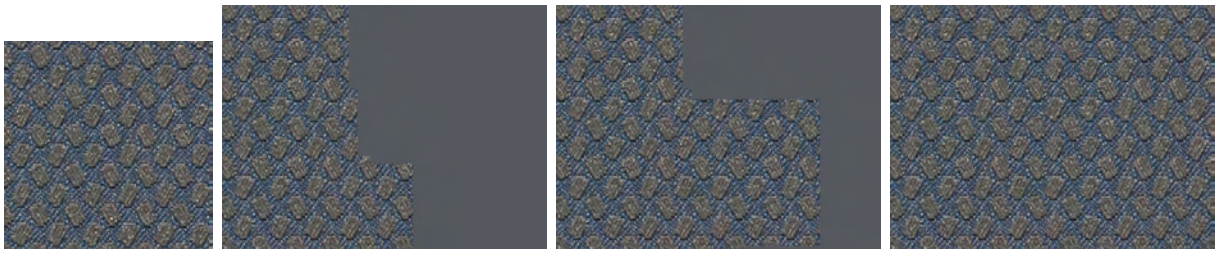



**Fig. 2.16: A new texture (right) is synthesized from an input sample (left) by iteratively copying selected parts and merging them into the intermediate result.**

layer of the output texture pyramid. Synthesis then proceeds pixel by pixel, layer by layer. Whenever a new pixel is synthesized, the copied pixel is selected randomly from a set of pixels from the input pyramid which have similar texture features (i.e., filter responses) at the current and all lower resolutions. The approach leads to much better results than the approach of Heeger and Bergen [212], demonstrating the effectiveness of the sampling method. Unfortunately, it spends a huge amount of time for determination of texels with matching features even if acceleration techniques like the one of Bar-Joseph et al. [20], which reuses results from lower resolution layers, are used.

A substantial improvement to the algorithm was introduced by Efros and Leung [123]. Instead of sampling randomly from a set of pixels with matching texture features, they propose to assign sampling probabilities based on similarity. As a similarity measure, they suggest a simple root mean square color difference of pixels in the neighborhoods of the current pixel and the candidate pixels. Optionally, the influence of pixels close to the center can be increased by scaling differences by a 2D Gaussian centered in the current pixel. The approach of Efros and Leung drops the idea of making decisions based on filter responses but instead uses pixel neighborhoods directly. Judging from the success of this method and following variants, this change seems to be a good choice.

Efros and Leung's method was later extended into several directions. One group of methods concentrated on the reduction of computation time by implementing more efficient algorithms for determining sets of matching candidates. Wei and Levoy [572] proposed to determine best matching candidates using tree-structured vector quantization. In a preprocessing step, pixel neighborhoods from the input texture are clustered iteratively, leading to a tree of clusters. During synthesis, well matching neighborhoods are quickly found by traversing the tree. Although this approach does not return best matching neighborhoods in all cases, the results look almost the same than using the much slower, brute force algorithm of Efros and Leung. A different strategy for determination of candidate sets was proposed by Ashikhmin [10]: He observed that for textures showing natural elements like flowers or grass better results can be achieved

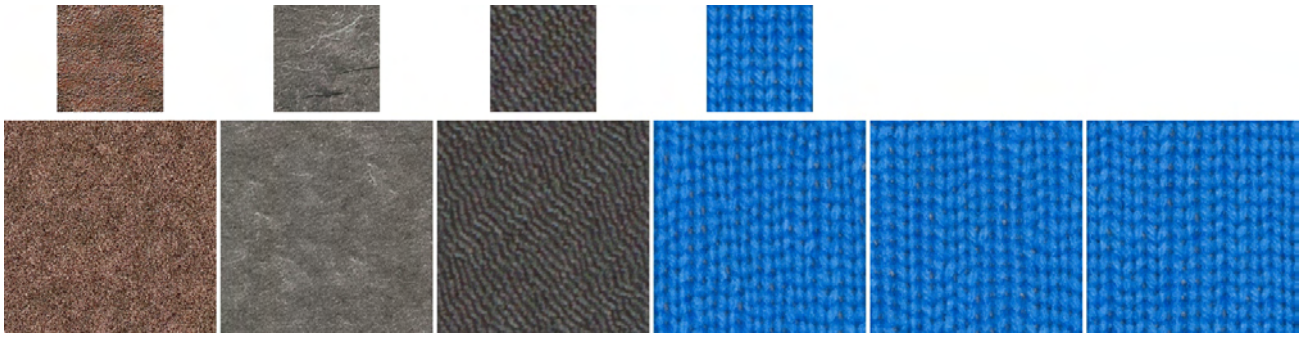


**Fig. 2.17: Synthesis results with pixel-based methods. While the algorithm succeeds in reproducing the textual elements, enforcement of regular structure fails even for large neighborhoods. Used neighborhoods from left to right: $7^2$, $7^2$, $4^2$, $7^2$, $10^2$, and $12^2$.**

by copying connected regions from the sample image. Therefore, he proposed to select candidates based on texture coordinates of synthesized pixels in the neighborhood. E.g., if the pixel copied to the left upper position of the currently processed pixel has texture coordinates $(15, 20)$ then the pixel with texture coordinates $(16, 21)$ from the input texture is considered a candidate. Tong et al. [520] combined these two approaches and simply employed the union of both candidate sets, leading to good results for a large number of textures. Zelinka and Garland [609] used a simpler, even faster method, capable of real-time texture synthesis. They copy subsequent pixels from the original texture and probabilistically jump to a different copy source location. Unfortunately, the probabilistic jumps may easily introduce well visible seems. Therefore, this approach is not suitable for high-quality texture synthesis.

A second area of improvement attacks the order-dependent synthesis results. Wei and Levoy [574] proposed an algorithm that iteratively improves an initial image by synthesizing pixels that match the neighborhood from the previous iteration. An especially interesting property of this approach is that it enables real-time implementations on GPUs [297] since pixels can be processed in parallel. The approach was improved in terms of quality by Kwatra et al. [287]. They introduced dependencies between neighboring pixels which guarantee that the quality of synthesized textures w.r.t. a similarity metric improves in each iteration. While this approach leads to clearly improved results, it reintroduces the necessity to compute texture synthesis in a central process, thus hindering efficient implementations on the GPU.

A third area of improvement concerns generalization to other types of data. Bar-Joseph et al. [20] generalized the approach of De Bonet [91] to handle 1D and 3D data (i.e., sound and movies). Instead of the steerable pyramid used for 2D data, they employed Daubechies wavelets for 1D data and a combination of Daubechies wavelets and a steerable pyramid for 3D data.

Due to the numerous improvements to pixel-based synthesis algorithms introduced in the last years, this class of methods was able to demonstrate its suitability for a huge number of textures (cf. Figure 2.17). Especially high-quality synthesis results and rather low processing times contributed to the success of these approaches. Unfortunately, experience showed that efficient processing prohibits similarity matching of large pixel neighborhoods, which are required to capture textures with large structures even if multi-scale synthesis algorithms are employed (cf. Figure 2.17). Therefore, the effectiveness of these approaches drops significantly especially for regular textures and those containing large elements.

The second class of image-based synthesis algorithms are the patch-based ones which copy connected regions instead of single pixels and combine existing and copied pixels as faithfully as possible. The key differences of existing algorithms are the way patches are selected from the source and placed in the target image, and the way they are combined. Efros and Freeman [122] select fixed size blocks and paste them in a regular manner such that the color differences in the overlap region are minimized. They combine the images using dynamic programming. A similar approach is followed by Liang et al. [311] but they apply feathering (i.e., blending) to merge overlapping patches. Kwatra et al. [288] generalize and improve these approaches by reducing the problem of combining the pixels to a maximum flow or minimal cut problem and allowing for blocks of arbitrary sizes. Regions of varying size are as well supported by the algorithm of Nealen and Alexa [376], which combines pixel- and region-based texture synthesis. A different approach is used by Xu et al. [600]: They start from a texture filled with tileable patches and then subsequently copy texture blocks, with the copy sequence, source and target positions defined by an almost chaotic transformation. To resolve mismatching pixels at block boundaries, pixels near the boundary are removed and resynthesized using fast pixel-based approaches. Another different but much more successful approach is the method of Dischler et al. [109]. It first applies a texture segmentation procedure to identify meaningful regions and then synthesizes textures based on spatial analysis of relative position of these regions in the original texture.

Patch-based texture synthesis algorithms are very successful for a large variety of textures. Figure 2.18 gives an overview of the capabilities of these algorithms, showing that they can handle all types of textures and that they achieve faithful results. Yet, despite the fact that copying large texture regions allows for preservation of large texture structures, the exact alignment of structures in the target texture often fails due to bad placement of patches. Especially faithful, simultaneous reproduction of structures at various scales is difficult, as the images in Figure 2.18 show.

To overcome this problem, the concept of *texture masks* was developed (cf. Figure 2.19). These masks steer or restrict the choice of candidates and are thus either used

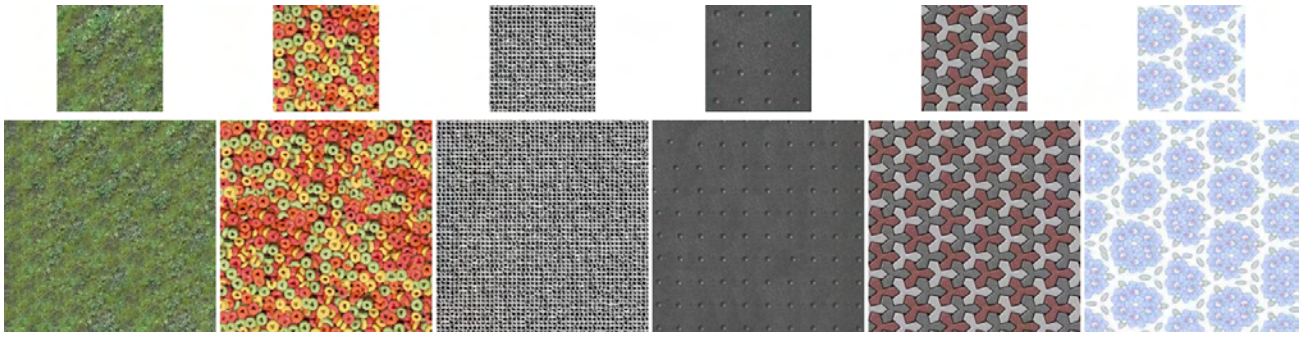


**Fig. 2.18: Synthesis results with patch-based methods. From left to right: stochastic, structured, two near-regular, and two regular textures. While the algorithm succeeds in reproducing the textual elements, enforcement of regularity sometimes fails.**

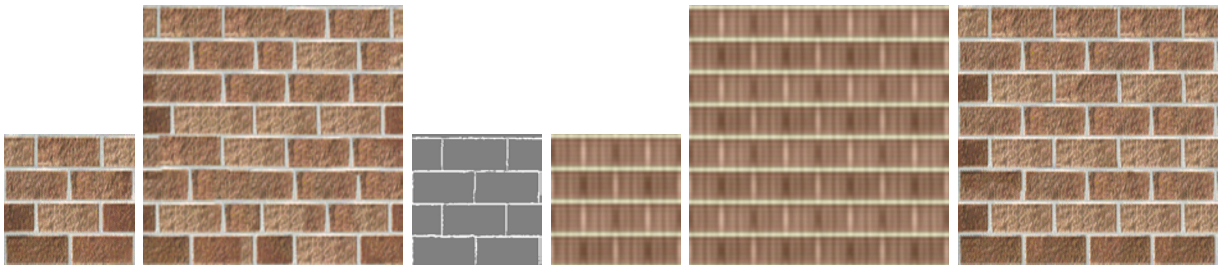


**Fig. 2.19: Texture Masks. From left to right: input texture, texture synthesized with patch-based algorithm showing broken features and missed regularity, texture mask extracted with method similar to [592], texture mask extracted with own method, enlarged texture mask, patch-based synthesis result including mask information featuring no broken features and correct regularity.**

to guide consistent texture placement [321] or to enforce that the shape of prominent features, e.g., from animal fur [611] or high-frequency edges [592] is not broken apart during texture synthesis. Unfortunately, determination of such texture masks is not trivial. Liu et al. [321] determine masks from height maps, which are usually not available for textures, and which are difficult to derive. Zhang et al. [611] require the user to specify such texture masks, which are restricted to binary values in their case. Wu and Yu [592] compute high-frequency features from input textures using feature detection filters. While being an automatic process, this often leads to undesired results due to the necessity of introducing thresholds to discriminate features, and due to the specific choice of suitable feature detection filters. Another approach for determination of feature masks is described in Chapter 4. As in the approach of Wu and Yu, masks are computed automatically. Unlike in Wu and Yu's approach, features with high energy are extracted, corresponding to the regular patterns of near-regular textures.

Another approach for handling regular and potentially large structures during texture synthesis was developed by Liu et al. [322] and Liu and Tsin [324]. They used the

observation that an infinite variety of periodic patterns can be characterized by a finite number of symmetry groups to extract tileable parts of a texture that correspond to the regular structure. Tiling these textures with overlap and merging the tiles leads to very good results for most regular textures. The approach was recently extended to handle near-regular textures [323] by treating them as deformations of regular textures. One drawback of this method is that extraction of tiles requires significant user intervention whereas methods based on texture masks are fully automatic. Another problem is that it requires at least two complete tiles to be included in the sample texture. Although this usually poses no problem, such data may be unavailable if regular structures of different scales are included in the texture. Finally, the approach may introduce visible seams in synthesized textures due to the merging process of neighboring tiles.

With all the extensions available for sampling-based texture synthesis algorithms, this class of texture synthesis algorithms can currently be considered the better choice compared to model-based approaches. Several algorithms are able to successfully handle a wide spectrum of textures and to generate very convincing results of arbitrary size. Unfortunately, synthesis times and storage requirements are directly proportional to the area of the synthesized texture, rendering the approach inappropriate in combination with large, textured areas. In such cases either combinations with other techniques like those described in the following, or application of parametric model-based techniques, which generate a model of the texture and thus easily allow for efficient generation of textures of arbitrary size on-the-fly, should be considered.

### 2.2.2 Texture Tiling

The methods for texture synthesis presented previously succeed in generating very good results for a large range of textures, i.e., they are highly successful at synthesizing textures with visual properties similar to the ones of an input texture. Unfortunately, such methods are often not sufficient for practical applications since they require storage of every synthesized pixel. If huge areas are to be covered this leads to huge storage requirements which can often not be afforded. A solution to this problem is synthesis of tileable textures which can be placed next to each other without visible seams, which makes the storage requirements independent of the area covered by the synthesized texture. If textures are large compared to the size of the structures they contain, resulting repetitions will hardly be visible.

To further reduce the probability of detecting repetition artifacts, researchers have proposed more sophisticated tiling approaches. Cohen et al. [73] proposed the use of Wang tiles, i.e., rectangles with an imaginary color-coding of edges where edges with identical colors imply a seamless match between the textures stored within the Wang

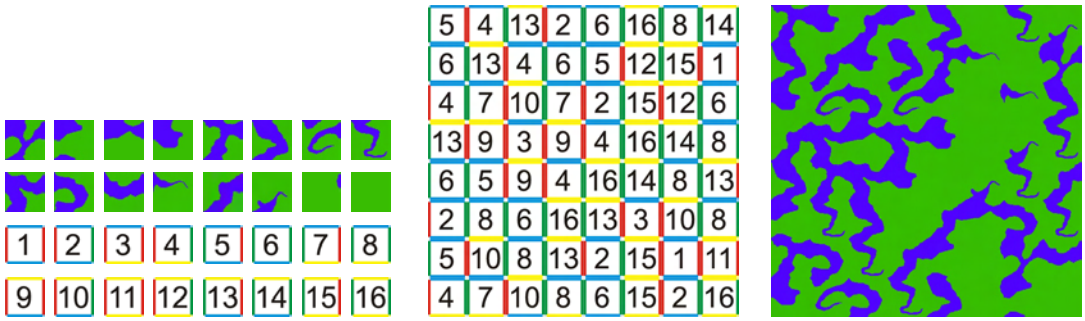



**Fig. 2.20: Synthesizing large textured areas (right) from a set of Wang tiles (left) by placing rectangular tiles with matching colors next to each other (middle).**

tiles (cf. Figure 2.20). Wang tiles lead to an aperiodic tiling of the plane if at any point during patch placement the user has the choice between at least two tiles. The color-coded edges additionally ensure visually rich textures since different patterns can be encoded. In their paper, Cohen et al. describe a method to compute such tiles from large input textures. A similar method based on single edge colors is described by Somol and Haindl [492].

### 2.2.3 Synthesis on Surfaces

Using a combination of surface parameterization, texture synthesis and texture tiling methods, it is possible to apply materials to surfaces with large areas. Unfortunately, problems may occur at each of these three stages (e.g., parameterization and synthesis algorithms might not be optimal for the given surfaces and textures, or surfaces might lack necessary preprocessing for the parameterization step). Therefore, alternative methods were proposed which achieve the same result in a single step. These methods apply material to surfaces by directly synthesizing material on the surfaces which can be thought of as a patchworking step, where parts of the material are cut out of a predefined patch and are glued to the surface. Obviously the synthesized material should appear to be a single continuous material patch, i.e., obvious material discontinuities like seams should be minimized (cf. Figure 2.21).

As for synthesis of 2D textures, respective methods can be divided into ones operating on the pixel level and those handling entire texture regions. Methods from the first category are directly derived from extending 2D synthesis methods to arbitrary 3D surfaces. The key problems include the definition of neighborhoods on 3D surfaces (which is commonly solved by considering connectivity) and the generalization of 2D methods from regularly sampled images to irregularly sampled surfaces (i.e., meshes). Initial solutions to these problems were proposed almost concurrently by Wei and Levoy [573],

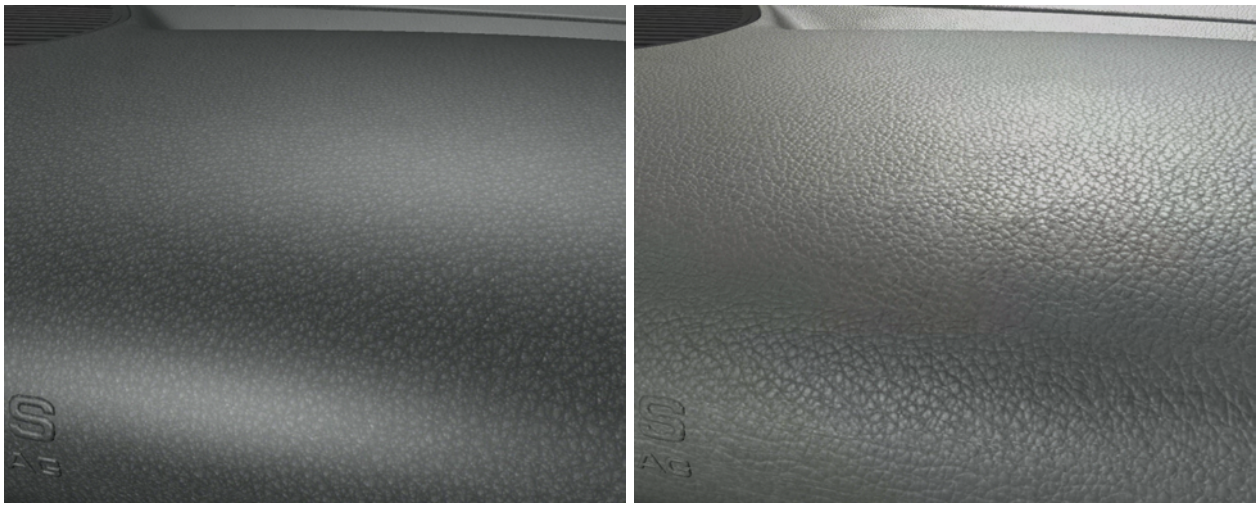


**Fig. 2.21: View of a surface textured with the approach of Magda and Kriegman [334]. No repetition is visible. At close inspection (cf. Figure 14.3), the diffuse texture (left) contains visible seams, which are more eye-catching if BTFs are synthesized and rendered (right).**

Turk [526], and Ying et al. [601]. All three approaches decide to synthesize texture colors at vertex locations and employ a hierarchical synthesis scheme. Having a sufficiently fine sampling of the surface, derived either with subdivision techniques [573, 601] or by distributing a sufficient number of points on the surface [526], these vertex colors can be resampled into a texture. Before synthesis starts, the methods require the definition of an orientation field to align the orientation of vertices with the 2D texture elements. Once this is done, vertex colors can be synthesized analogously to pixels in the 2D case: For each vertex, a neighborhood is established based on connected vertices, the neighborhood is flattened, and the colors from this neighborhood are resampled into a regular structure. A vertex color can then be chosen according to any pixel-based 2D synthesis scheme. These techniques achieve good results for a wide variety of textures but the quality of texturization largely depends on the specified direction field. Especially problematic cases occur at singularity points of direction fields (e.g., poles of spheres).

Possibly linked to the superior synthesis quality of patch-based methods in the 2D case, more synthesis on surfaces methods from the second category, i.e., handling patches, were published. An initial method was proposed by Neyret and Cani [381], who employ an approach similar to Wang tiling [73] but based on equilateral triangles with imaginary edge-colors and possible corner constraints. The authors propose an automatic and a manual method for construction of the tile set but are limited to isotropic textures. Additionally, they require meshes to feature approximately equilateral triangles, which is often not the case and thus requires complex retriangulation. A more advanced method handling anisotropic textures was presented by Praun et al. [421]. Predefined surface patches are mapped onto surface patches, thereby optimizing tex-

ture orientation and scale (which are derived from an orientation field). Surface patches are determined by a region-growing approach starting at random seed triangles. Badly matching borders of texture patches are hidden using alpha-blending, which can either be performed during rendering or the blended results can be stored in a texture atlas. Unfortunately, this simple approach turns out to be insufficient for structured textures.

A significant improvement was introduced by Soler et al. [491] who did not limit texture patches to a predefined set of textures but who chose best matching patches from an input texture. In addition, they introduce the idea of splitting surface patches if the error resulting from texturing the whole patch becomes too large, which is similar in spirit to the texture synthesis algorithm of Nealen and Alexa [376]. Another improvement was proposed by Magda and Kriegman [334] who reduced time requirements for determining best matching patches: Instead of matching the color values in regions along the edges, they compare 2D textons (i.e., clusters of responses to oriented filters [304]) along the edges, reducing the problem from 2D to 1D. Precomputing distance values of textons achieves significant speedups, allowing them to handle triangles individually, resulting in finer control than handling patches of triangles. This leads to better results for highly structured textures since the texture orientation can vary per triangle. An example image showing the result of texturing a surface with this approach is the left image of Figure 2.21. Another approach targeting even faster synthesis was published by Zelinka and Garland [610]. Fast searches for matching, triangular patches are realized by extending the search strategy of Ashikhmin et al. [10] to meshes, which requires floating point instead of integer offsets. Their approach establishes matching textures at edges by assigning connected parts from the texture to neighboring triangles. While being a very fast method, the results of this approach are significantly worse than the ones from Magda and Kriegman.

Another group of texture synthesis on surfaces algorithms intelligently combines suitable parameterization techniques and standard 2D texture synthesis approaches. Fu and Meung [154] parameterize arbitrary objects using polycube textures [515] and apply textures to surfaces by assigning Wang tiles to sides of the polycubes. Wang et al. [557] propose an approach based on conformal parameterization [306]. To compensate for the missing area preservation of conformal maps, which leads to differently scaled texture regions on the mesh, they synthesize textures specifically for a parameterized mesh, taking into account local texture stretch (i.e., textures are synthesized at larger scales in regions with high stretch and at smaller scales in regions with low stretch). The combination of both methods leads to textures that seem to be mapped according to an isometric (i.e., area- and angle-preserving) parameterization. The authors propose an efficient implementation of the synthesis step based on patch-based synthesis [288] from multiple textures, representing the sample texture at various scales. Unfortunately,

like all methods based on parameterization, the approach has to deal with texture cuts. Additionally, synthesized textures need to be stored in a texture atlas, which requires large amounts of memory, and, due to efficiency reasons, the synthesized textures result from discrete scales only.

Applying textures directly to surfaces using synthesis on surfaces techniques has become very popular recently. Resulting, textured models show hardly any cuts and require few storage. In addition, almost arbitrary models can be handled easily, since these methods require local parameterizations only, which are much simpler to compute than global parameterizations. This simplicity is reflected by the appearance of simple and efficient tools that enable replacement of materials in photographs by extracting height-field geometry and performing synthesis on surfaces [137, 607].

Obviously, these methods have drawbacks as well. Compared to 2D texture synthesis methods, much more efforts have to be spent on handling the irregularity of meshes. This is mirrored by the fact that synthesis of highly structured or regular textures on surfaces is still a very complex problem. In addition, textures synthesized on surfaces need to be regenerated completely if the scale, orientation, or type of texture is exchanged. Additionally, such textures are valid for a single object only. Applying combinations of global parameterization, texture synthesis and texture tiling techniques, these restrictions do not apply.

### 2.2.4 BTF Synthesis

Previously techniques for enlarging material samples represented by diffuse textures were described. Obviously, many more spatially varying material representations exist. Therefore, the question is whether these texture techniques generalize to arbitrary reflectance properties.

In principle, synthesis techniques do not depend on the data they handle. They model a stochastic process that generated the input sample independently of the data it represents. Therefore, handling of other spatially varying data like height or bump maps works exactly the same way as synthesis of color textures [296, 321].

Slight changes to the algorithms have to be introduced when handling reflectance data of much higher dimensionality like tabulated BTFs. Due to the large amount of data per pixel, existing synthesis techniques become infeasible. A first approach for handling BTF data was published by Liu et al. [321]. They first estimate the height field of the material using shape-from-shading techniques and synthesize a larger version of it. Then, for synthesizing a specific slice of the BTF (i.e., fixed view and light direction), they approximate the appearance of the synthesized material patch given the current view- and light direction, the synthesized height field, and assuming an average BRDF.

The resulting image is used as an initial image for a patch-based synthesis technique. A problem of this technique is individual handling of BTF slices which may easily lead to inconsistent shading when changing light or view directions.

A different approach was introduced by Tong et al. [520]. They proposed an efficient algorithm based on 3D textons [304], which are derived by applying filter banks to (a subset of) the BTF slices, concatenating the filter responses, and performing k-means clustering of the resulting vectors. Distance values between textons are precomputed, requiring simple lookups during the pixel-based BTF synthesis procedure only. Obviously, this is much faster than computing similarities of ABRDFs. Unfortunately, the results may show significant clustering artifacts which get worse if less clusters or subsets of the BTF slices are used. This problem was resolved by the approach followed by Koudelka et al. [278] and Liu et al. [319]. They reduce the dimensionality of the BTF by retaining only the most significant, spatially dependent reconstruction weights from factorizing the BTF into spatially dependent and angularly dependent parts. Since these coefficients have linear influence on the reconstruction, they can simply be used with the standard similarity metrics instead of color values. While the authors used this approach for pixel-based synthesis, it can obviously be employed with any material synthesis algorithm. The right image in Figure 2.21 shows a surface textured with BTF using the approach in combination with a synthesis on surfaces technique [334].

## 2.3 Discussion

The previous sections presented existing approaches for making accurate material representations available for modeling of virtual scenes and rendering. Due to the large number of material representations and material synthesis methods, some specialized, some very general, no clear choice of an optimal combination of methods for all kinds of application areas can be made.

Concerning material representations, tabulated BRDFs currently provide the most accurate rendering results since BRDF acquisition devices are well calibrated and since BRDFs can be sampled for a dense set of directions. Unfortunately, BRDFs fail to reproduce complex materials which show significant spatial variation due to meso-scale structures or spatial material changes. Combinations with bump- or displacement-mapping techniques can resolve these problems to some extent but typically miss important local effects like subsurface scattering, self-shadowing, or interreflections. All these effects can efficiently be reproduced with BTF representations. Like BRDFs, BTFs can be acquired using automated setups. Typically the angular resolution of measurements is well below the one for BRDFs and existing setups are less well calibrated. Since these limitations are not of principal nature, they will most likely be removed in the future, making BTFs the currently best choice for modeling of general materials. A significant contribution to this success can be attributed to the developed BTF compression techniques, which make handling of the huge amounts of data from BTFs possible. In the following chapter, one such technique based on reflectance fields is presented, which achieves very high compression rates while concurrently enabling high-quality data reconstruction at real-time frame rates.

Despite the accuracy of existing material representations, no material datasets suitable for truly predictive rendering have been specified to date. Existing measurement setups for BRDFs or BTFs only handle a small number of spectral bands (typically only RGB measurements), prohibiting spectral rendering, which is essential for predictive rendering. Additionally, many acquisition setups feature a limited dynamic range, often restricting color values to 8 bits. In the future, it will be absolutely necessary to remove these restriction, leading to significantly more accurate but also more complex material data. Once such data is available, existing compression techniques will be extended to these more complex data sets.

The second part of this chapter focused on existing techniques for material synthesis. Again, the huge amount of available methods makes it difficult to indicate a single, best method. Judging from their recent popularity and the quality of synthesized materials, patch-based, intelligent sampling techniques can currently be considered the methods of choice for a wide variety of textures. Existing extensions of these techniques based on

texture masks extend the application areas to almost all kinds of materials. In Chapter 4, a technique for automatic generation of such texture masks is described which allows patch-based synthesis methods to handle regular and near-regular materials.

Although patch-based sampling techniques appear to be the superior material synthesis techniques, they cannot be considered optimal. As shown above, in practice texture tiling or synthesis on surfaces techniques are required to texture objects with large surfaces without introducing notable repetition artifacts. For synthesizing 2D textures, other approaches feature significant advantages (i.e., pixel-based sampling techniques allow for finer control of the synthesis process, and methods based on parametric texture models achieve much more compact texture representations). Therefore, future trends for texture synthesis techniques are difficult to predict. A definite goal is the development of highly accurate parametric MRF models, which would combine the accurate texture reproduction capabilities of existing patch-based sampling methods with the compact representation of parametric texture models. Optimally, such methods would be efficient to evaluate, allowing for real-time synthesis on the GPU whenever accessing a specific texture element. Clearly, these techniques need to take into account the increasing realism and complexity of available material representations, and possibly it will be necessary to extend them to volumetric data for handling light-matter interaction effects like subsurface scattering.

# Chapter 3

# Reflectance Field based BTF Compression

In the previous chapter, several existing material representations were reviewed, resulting in the assessment that currently measured BTFs are the most accurate general-purpose material representation. Therefore, they qualify for demanding applications like predictive rendering. The most significant problem with these representations is their huge memory requirements, typically several GBs for BTFs with quite limited spatial extent, especially if view and light directions are sampled densely. To resolve this problem, data compression techniques become inevitable. Obviously, these techniques need not only reduce the storage requirements but should additionally allow for accurate reconstruction, optimally at real-time frame rates. Additionally, they should be easily implementable on existing GPUs to allow for efficient integration into existing real-time graphics APIs.

In the following, a compression technique fulfilling these requirements is described. It was first presented in [358], a later extension is described in [359]. For completeness the description starts with an overview of the BTF measurement process acquiring the data which is later compressed.

## 3.1 Measured BTF Data

The BTF measurement setup used to acquire the BTFs shown in the following was first described in [206] and is representative for a number of setups used all over the world (a more complete description can be found in [335]). It is composed of four main parts: a lamp with fixed position approximating directional sunlight, a programmable high resolution CCD-camera mounted on a wagon which resides on a rail system forming a

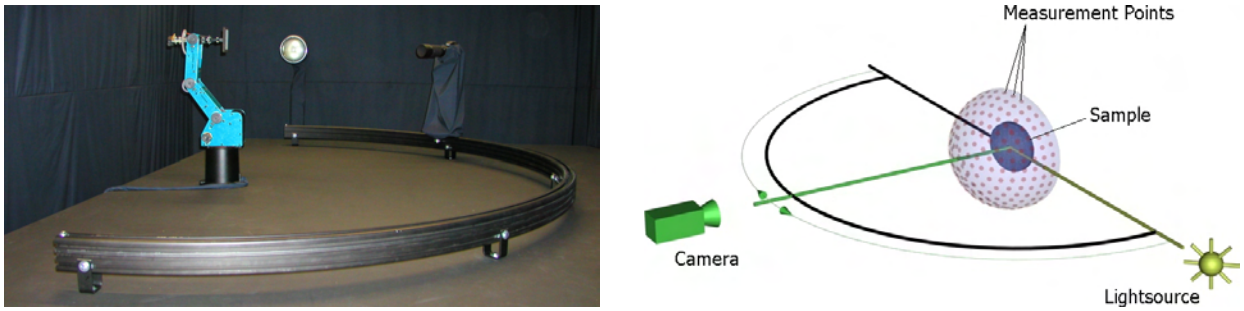



**Fig. 3.1: BTF measurement laboratory. Left: photograph. The black casings of the robot and light source are removed. Right: scheme showing the fixed light source, moving camera, and the sampled directions as points on the sample's hemisphere.**

half-circle around the probe holder, a robot arm holding the probe with a maximum extent of 10 cm $\times$ 10 cm, and a personal computer (cf. Figure 3.1). To reduce the influence of scattered light, the lab is darkened and the hardware used for measurements is covered with dark casings. This computer controlled setup allows automatic reproduction of every constellation of view/light-direction with reference to the sample surface.

To achieve accurate measurement quality, several calibration steps are performed, aiming at compensating the positioning error due to the robot and the railsystem, minimizing the geometric calibration error due to the optical system of the camera, and optimizing color reproduction. Acquisition quality is optimized by sampling a dense number of points on the hemisphere of light and view directions (81 almost equally spaced directions each, resulting in a number of 6561 photographs), and capturing high-dynamic range images using the technique of Debevec and Malik [95]. For practical reasons, only RGB measurements are taken, which limits the accuracy of the data compared to spectral measurements significantly.

Storing the 6561 images for a single exposure time of the HDR series requires about 80 GB, 12 MB per image, employing the Kodak DCR 12-bit RGB format and lossless compression. Taking a typical HDR series of four exposure times, this leads to a total amount of 320 GB. Data size is reduced significantly during the data postprocessing step already (cf. Figure 3.2): Images are rectified, regions of interest are cut out, and the cut images from various exposure times are converted into 16 bit float HDR images. These steps reduce the storage requirements to about 1.5 GB for BTFs with a spatial extent of 256 $\times$ 256 pixels, which is obviously far too much for efficient rendering on graphics boards, especially if several materials are contained in rendered scenes. Therefore, BTF compression is absolutely necessary for efficient handling of this data.

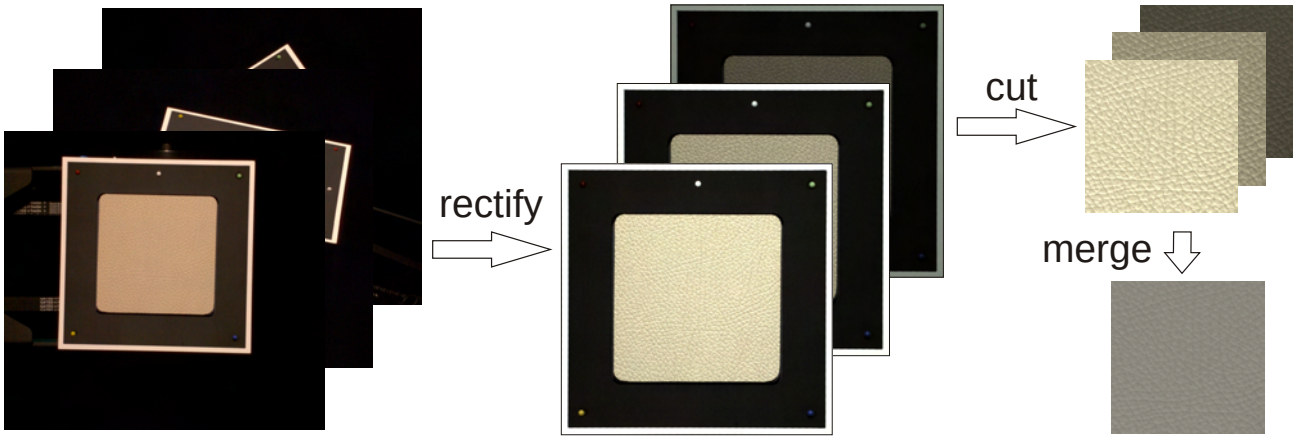



**Fig. 3.2: Postprocessing Pipeline. The left part shows sections from taken images. The images are rectified, cut, and converted into a HDR BTF representation.**

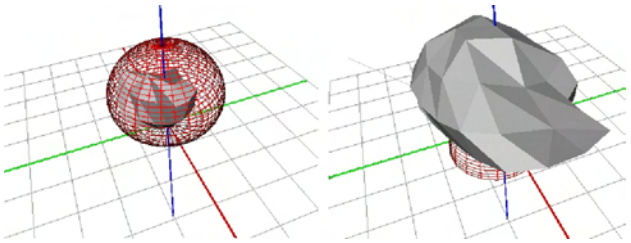


**Fig. 3.3: Plot of original (solid) and fitted data (wireframe) for a knitted wool BTF texel (cf. Table 3.1) and two view directions.**

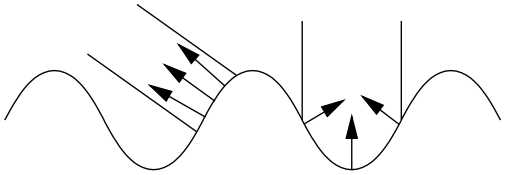


**Fig. 3.4: Changing perceived normals for varying view directions.**

## 3.2 BTF Compression

As mentioned in the previous chapter, two principally different approaches to BTF compression exist. On the one hand, researchers [348] fitted parametric BRDF models to BTF texels, leading to tremendous compression ratios and fast rendering algorithms. On the other hand, techniques based on data-driven compression were developed, which reduce the dimensionality of the data by, e.g., applying singular value decomposition and retaining the most important components only. Techniques from the second category typically achieve better reconstruction quality than parametric BRDF models but since a large number of components needs to be retained, compression rates and reconstruction speed are inferior. Due to the necessity to achieve real-time frame rates in VR applications, in the following a technique from the first category is presented.

Fitting BRDF models to every texel of a BTF is a straightforward way of compressing BTFs. Nevertheless, the success of such approaches is limited since BRDF models assume the modeled data to be reciprocal and energy conserving. This assumption does not hold for per texel BTF data due to the two following reasons:

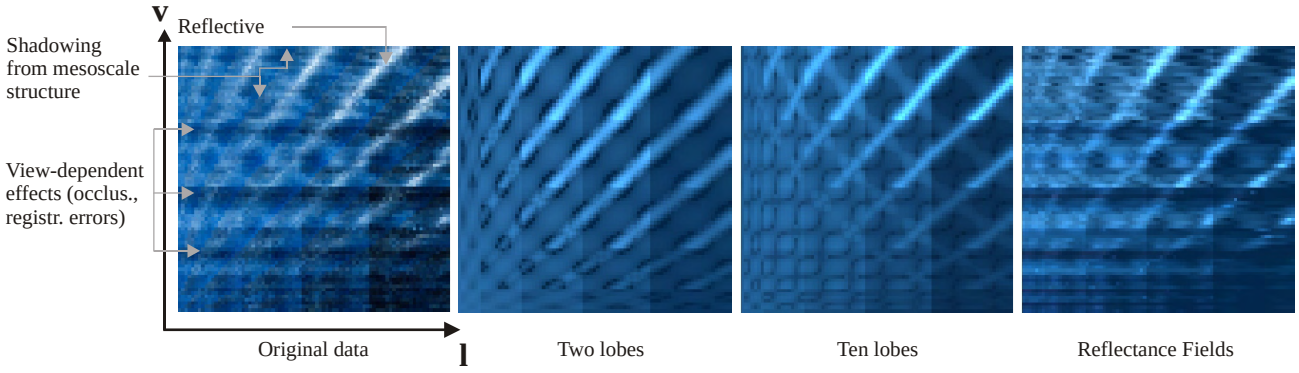



**Fig. 3.5: ABRDFs for original (left) and fitted data of knitted wool BTF (cf. Table 3.1). Note, the foreshortening term $\mathrm{n} \cdot \mathrm{l}$ is included. Even ten Lafortune lobes are not capable of capturing the asymmetry in the data which is improved significantly when fitting reflectance fields.**

- BTFs feature asymmetric effects due to subsurface scattering, self-shadowing, and self-occlusion. Therefore, per texel values are not reciprocal like BRDFs.

- The physical motivation of analytic models relies on the fact that the underlying microgeometry of the surface can be modeled with a certain distribution of microfacets, which is modulated by a shadowing and a Fresnel term. The shadowing term of simple cosine-lobe models is insufficient to express the discontinuous shadowing and masking effects taking place on rough surfaces (cf. Figure 3.3). In particular, they are not capable of modeling the view-dependent effect of changing normals which is experienced for materials with significant mesostructure (cf. Figure 3.4).

These effects can well be observed in Figure 3.5 which shows the ABRDF of one texel of the knitted wool BTF (each pixel represents the ABRDF value measured for one combination of light directions $\mathbf{l}$ and view directions $\mathbf{v}$). Clearly, the diagonal reflective peeks are visible whenever the view direction equals the mirrored light direction. Additionally, self-shadowing and self-occlusion effects due to surrounding mesostructure are visible as dark regions. Fitting standard Lafortune [290] BRDF lobes to the data cannot capture these features, even if a large number of lobes is employed.[1]

To better capture the properties of per texel data, Daubert et al. [89] introduced an additional view-dependent scaling term, which can be understood to model view-dependent self-occlusions of the material. This modification improves reconstruction quality for materials with minor mesoscale structure as the wallpaper in Figure 3.6. Yet,

[1]Note: Fitting a large number of Lafortune lobes is also prohibitive in terms of processing time and accuracy since it requires time-consuming, instable, non-linear optimization procedures, leading to run-times of several hundred hours for a $256 \times 256$ texel BTF.

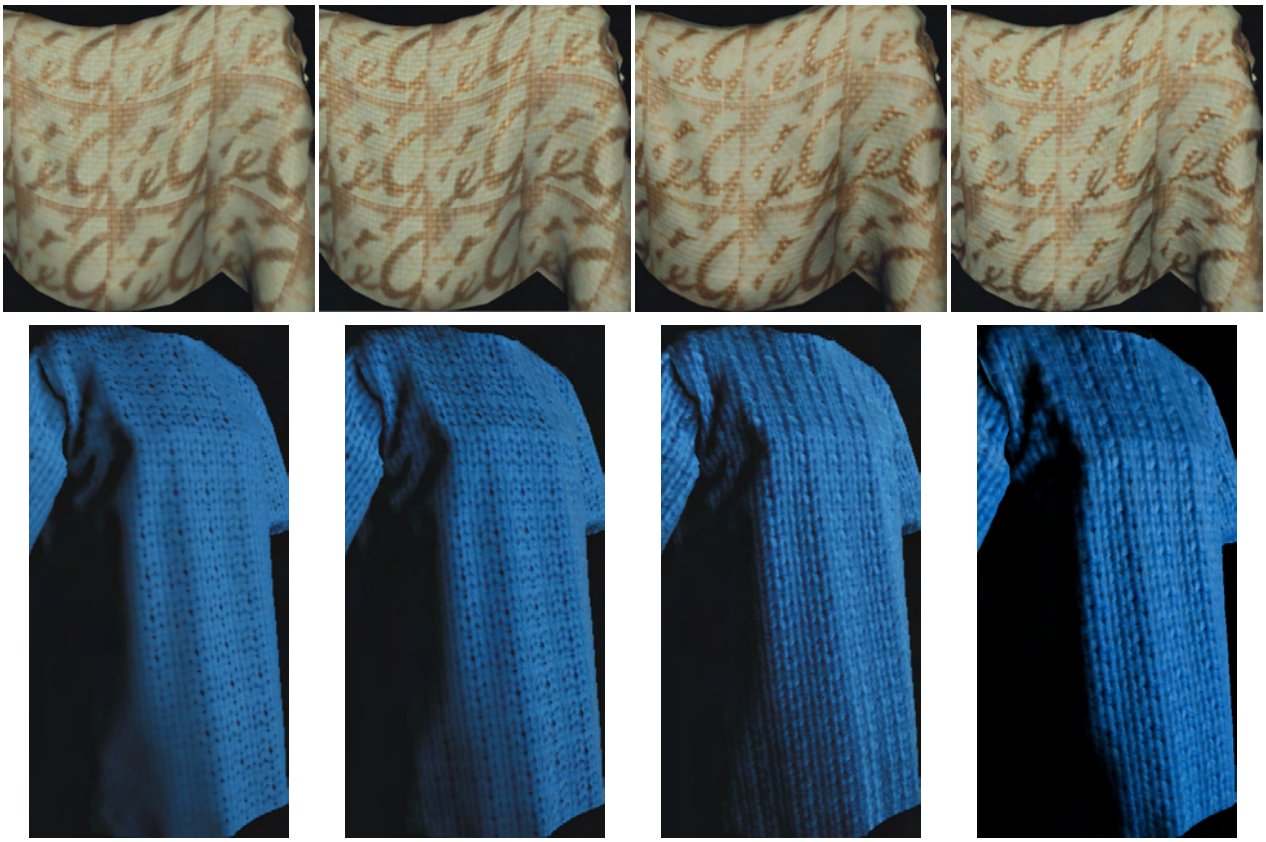


**Fig. 3.6: Suitability of per-texel ABRDF models for compressing BTFs with low height variation. Top: wallpaper material, bottom: knitted wool. From left to right: Lafortune lobes [348], scaled Lafortune lobes [89], RF based, and high quality reconstruction [447].**

it is not sufficient for modeling the substantial self-shadowing and self-occlusion effects commonly observed for materials with significant mesoscale structure as the knitted wool in Figure 3.6. Obviously much better results could be achieved by (non)linear interpolation of the measured data, but obviously this is prohibitive in terms of storage requirements.

Therefore the idea of the compression techniques presented here is to determine an optimal combination of linear interpolation of data values, which results in highly accurate reconstructions, and parametric data models, which gives excellent compression rates. A suitable approach to this combination is avoidance of interpolation in both angular domains, leaving the choice to either interpolate light or view directions.

Fixing the incident direction $\mathbf{l}$ of a measured BTF dataset yields a 4D function called the *surface light field* (SLF):

$$\mathrm{SLF}_{\mathbf{l}}(\mathbf{x}, \mathbf{v}) := \mathrm{BTF}(\mathbf{x}, \mathbf{l}, \mathbf{v}).$$

SLFs have been used for 3D-photography, enabling the rendering of novel views of real-world objects under complex but fixed illumination [586, 62].

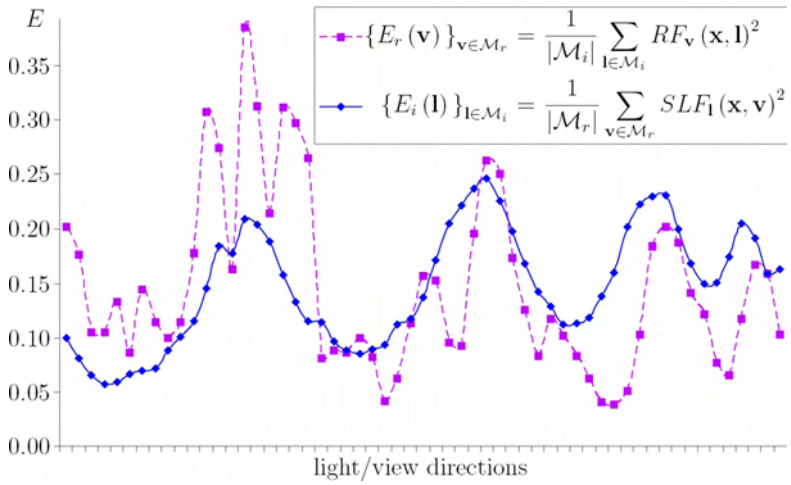



**Fig. 3.7: Plot of the $E_r$, $E_i$ series for one texel of the corduroy data set (cf. Table 3.1).**

Otherwise, fixing exitant direction $\mathbf{v}$, results in the patch's *reflectance field* (RF)

$$\mathrm{RF}_{\mathbf{v}}(\mathbf{x},\mathbf{l}) := \mathrm{BTF}(\mathbf{x},\mathbf{l},\mathbf{v}).$$

As expected, RFs are used for rendering of real world objects under a fixed view with arbitrary illumination [209] and allow for very compact representations [338] in the case of mainly diffuse reflection.

Choosing either representation, BTF rendering can be interpreted as mapping and rendering a discrete set of SLFs or RFs of the measured surface onto arbitrary geometry. Employing either representation, the color of a BTF-textured surface element with texture coordinate $\mathbf{x}$ given local light and view direction $(\mathbf{l},\mathbf{v})$ can be computed as follows:

- Approximate the BTF by a set of independently fit SLFs/RFs.
- Compute the color of $\mathbf{x}$ according to every fitted SLF/RF and interpolate the final color from the partial results.

## 3.3 Reflectance Field BTF Approximation

Several tests with both RF and SLF approximations determined that the reflectance field is better suited for approximation in the given approach. This can be explained by inherent BTF asymmetry, which will be discussed in the following.

Figure 3.7 depicts a plot of the energy $E$ contained in a single texel in each SLF or RF of the corduroy BTF, where $\mathcal{M}_i$, $\mathcal{M}_r$ denote the sets of measured light/view directions respectively. Note, that both series would be identical in the case of a reciprocal BRDF

at the texel (and $\mathcal{M}_i = \mathcal{M}_r$). In the given, typical case $E_r$ exhibits much more discontinuities (arising from depth variation) than $E_i$ – especially for grazing angles. This can be explained by two main reasons:

- First, interreflections and subsurface-scattering from neighboring texels and the fact that it is not possible to build a purely directional light source for the measurement process let the light act like a low-pass filter on the surface structure. In contrast, the small aperture of the camera provides a fairly directional view direction.

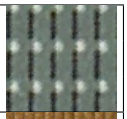
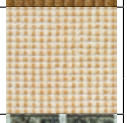
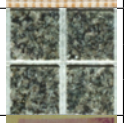
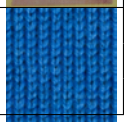
- Second, due to the rectification process the images of a SLF are convolved with different, view-dependent smoothing kernels of varying size. This increases the light-integration domain even further.

As a results, changes in light direction are smoother than changes in view direction. This was also noted by Malzbender et al. [338] in the case of mainly diffuse surfaces. Therefore RFs are better suited for fitting by compact functional representations (e.g., polynomials) than SLFs. The discontinuous view-dependence will be preserved by this approach, since the piecewise linear function as induced by the linear interpolation captures the high-frequency content of the data. Fortunately, RF approximations are favorable due to another, practical reason: Since there is only one viewer at rendering time, interpolation between the RFs has to be done only once per texel. Interpolating light-directions would require per light-source interpolation.

An implementation of this approach can be obtained by applying view interpolation and a suitable RF approximation, which should be efficiently renderable on existing consumer graphics hardware and minimize the approximation error. At a first glance, biquadratic polynomials [338] appear to be a suitable candidate, since they can be fit very efficiently and since reconstruction on graphics hardware can efficiently and easily be implemented. Another candidate are the non-linear functions

$$\begin{aligned} \mathrm{RF}_{\mathbf{v}}(\mathbf{x},\mathbf{l}) &\approx \rho_d(\mathbf{x}) + \rho_{s,\mathbf{v}}(\mathbf{x}) \sum_{j=1}^{k} s_{\mathbf{v},j}(\mathbf{x},\mathbf{l}) \\ &= \rho_d(\mathbf{x}) + \rho_{s,\mathbf{v}}(\mathbf{x}) \sum_{j=1}^{k} \left(t_{\mathbf{v},j}(\mathbf{x}) \cdot \mathbf{l}\right)^{n_{\mathbf{v},j}(\mathbf{x})} \end{aligned} \tag{3.1}$$

with $t_{\mathbf{v},j}(\mathbf{x})$ being a three dimensional vector and $s_{\mathbf{v},j}(\mathbf{x},\mathbf{l})$ similar to a Lafortune lobe discarding the exitant direction, $\rho_d$ and $\rho_s$ denoting diffuse and specular albedo. This model is better suited for fitting specularities and directional diffuse lobes than the biquadratic polynomials but requires more complex, non-linear fitting (e.g., using the

| | | LAF | SLAF | RFP | RFNL | |
|---|---|---|---|---|---|---|
| aluminium | avg | 0.0572 | 0.0558 | 0.0846 | 0.0479 |  |
| | min | 0.0391 | 0.0389 | 0.0773 | 0.0324 | |
| | max | 0.0935 | 0.0889 | 0.0985 | 0.0782 | |
| corduroy | avg | 0.1114 | 0.0859 | 0.0513 | 0.0537 |  |
| | min | 0.1003 | 0.0761 | 0.0416 | 0.0437 | |
| | max | 0.1223 | 0.0948 | 0.0604 | 0.0639 | |
| proposte | avg | 0.0978 | 0.0794 | 0.0689 | 0.0736 |  |
| | min | 0.0645 | 0.0537 | 0.0514 | 0.0490 | |
| | max | 0.1149 | 0.1003 | 0.0829 | 0.0924 | |
| stone | avg | 0.0904 | 0.0816 | 0.0797 | 0.0640 |  |
| | min | 0.0368 | 0.0345 | 0.0523 | 0.0284 | |
| | max | 0.1847 | 0.1767 | 0.1335 | 0.1321 | |
| wallpaper | avg | 0.0605 | 0.0565 | 0.0566 | 0.0455 |  |
| | min | 0.0380 | 0.0372 | 0.0420 | 0.0322 | |
| | max | 0.0983 | 0.0889 | 0.0772 | 0.0674 | |
| knitted wool | avg | 0.0788 | 0.0693 | 0.0719 | 0.0582 |  |
| | min | 0.0570 | 0.0527 | 0.0592 | 0.0459 | |
| | max | 0.1056 | 0.0875 | 0.0857 | 0.0729 | |

**Tab. 3.1: $\varepsilon$ for BTFs approximated by four Lafortune lobes per texel (LAF) and additional view-dependent scale factor (SLAF), RFs using biquadratic polynomials (RFP), and using two non-linear lobes (RFNL).**

Levenberg-Marquardt algorithm). Convergence is improved via detecting principal directions with a high-pass filter and using the recovered directions as an initialization for the optimization. The parameter $k$ controls the number of lobes.

## 3.4 Comparison with Previous Techniques

To measure and compare the approximation quality of the RF model quantitatively the average reconstruction error per texel $\mathbf{x}$ can be used, which is computed as follows:

$$\varepsilon^{B}_{\mathrm{BTF}}(\mathbf{x}) = \sum_{(\mathbf{v},\mathbf{l})\in\mathcal{M}_r\times\mathcal{M}_i} \frac{|\mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) - B(\mathbf{x},\mathbf{v},\mathbf{l})|}{|\mathcal{M}_i|\cdot|\mathcal{M}_r|} \tag{3.2}$$

$B$ denotes the corresponding BTF-approximation. The proposed model is compared to the Lafortune model for four lobes (higher numbers did not lead to significantly better results but increased the fitting times drastically as mentioned above), and the more sophisticated, asymmetric model of Daubert et al. [89] (cf. Table 3.1). As the comparisons show, the model based on fitting RFs achieves clearly improved reconstruction quality compared to the (scaled) Lafortune lobes. As expected, the RF method is especially

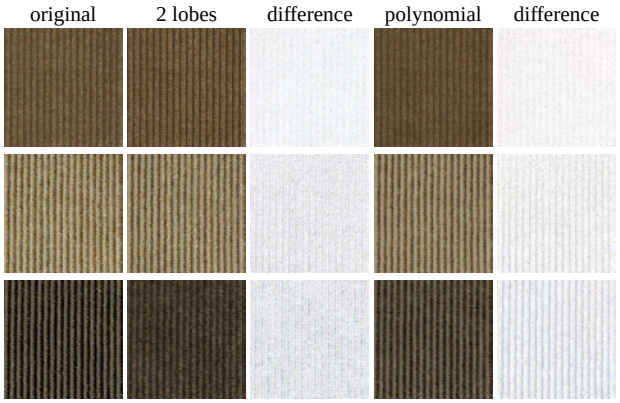



**Fig. 3.8: Comparison of the fitting methods: results for different view directions. While the two lobe model generates some specular highlights, the biquadratic polynomial shows the grazing angle problem.**

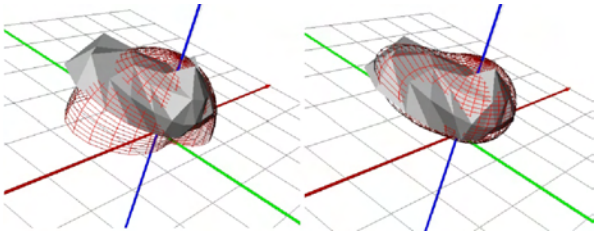


**Fig. 3.9: Plot of RF (solid) and fit (wireframe, left: biquadratic polynomial, right: 2 lobes) for one texel. Polynomials have problems at grazing angles while the summed lobes fit the data well.**

suited for depth varying materials like corduroy while flat materials like wallpaper and especially aluminium are also well approximated by simpler models. The comparisons additionally indicate that polynomial RFs work well for mainly diffuse materials like proposte or corduroy. For materials with strong specularities like wallpaper or stone the non-linear lobes are favorable since the polynomials tend to blur specularities (cf. Figures 3.8 and 3.9 for visual comparisons of fitting errors). Finally, the tests also revealed that $k = 2$ lobes are typically sufficient for generating satisfying results (cf. Figure 3.6 for results of the 2-lobe fit).

Interestingly, computation times for fitting RF lobes are comparable to fitting Lafortune lobes since more lobes have to be fit to less data. Total preprocessing times for a BTF with $256 \times 256$ pixels and $|\mathcal{M}_i| = |\mathcal{M}_r| = 81$ are about 32 hours if RFs are fit, and about 50 hours for Lafortune lobes.

Compared to the other two approaches, RF approximation requires significantly more memory. While the method of McAllister et al. [348] compresses a $256 \times 256$ pixels to about 1.4 MB using two lobes, the method of Daubert et al. [89] requires 31.8 MB already. In comparison, the RF method requires about 190 MB. Thus, this amount of data has to be reduced further to enable simultaneous rendering from several BTFs using graphics hardware.

## 3.5 Clustered Reflectance Fields

Since the amount of memory required for storage of RF compressed BTFs is directly related to the number of BTF texels (i.e., ABRDFs), an obvious approach for reducing

**Procedure 1** fitClusteredRFs$(\mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}),n)$

$\mathcal{S} \leftarrow \{\mathrm{ABRDF}_{\mathbf{v},\mathbf{l}}\} = \{\mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l})\}_{\mathbf{x}}$
$C \leftarrow$ kMeansClusters$(\mathcal{S},n)$
**for all** $\mathbf{x}$ **do**
  $I(\mathbf{x}) \leftarrow \mathrm{argmin}_{i=1\ldots|C|}\{C_i(\mathbf{v},\mathbf{l}) - \mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l})\}$
**end for**
**for** $j = 0\ldots|C|$ **do**
  **for all** $\mathbf{v} \in \mathcal{M}_r$ **do**
    $\mathrm{RF}_{\mathbf{v},j}(\mathbf{l}) \leftarrow$ fitRF$(C_j(\mathbf{v},\mathbf{l}))$
  **end for**
**end for**
**return** $I, \{\mathrm{RF}_{\mathbf{v}_i,j}(\mathbf{l})\}_{j=1\ldots n,\mathbf{v}_i\in\mathcal{M}_r}$

memory requirements is clustering of similar ABRDFs. E.g., representing a $256^2$ pixels BTF by 256 ABRDF clusters (each containing $|\mathcal{M}_r| = 81$ RFs) leads to a reduction from about 190 MB to about 0.9 MB, and for 1024 clusters to about 3.1 MB.

Pursuing this approach, the method for compressing BTFs changes to the one shown in Procedure 1: Initially, all the ABRDFs of the BTF are gathered in set $\mathcal{S}$. Then, k-means clustering [325] of the ABRDFs is performed, generating $n$ clusters that optimally represent the original data. The clusters are stored in a list $C$. Next, each ABRDF of the BTF is assigned the best fitting cluster and the index of this cluster is stored in an index map $I$. Finally, for each cluster center ABRDF and each sampled view direction, a (non-linear) reflectance field is fit to the corresponding BTF slice. The procedure returns the list of fitted RFs and the index map $I$. BTF values can be reconstructed from the returned entities as follows:

$$\mathrm{BTF}(\mathbf{x},\mathbf{v_i},\mathbf{l}) \approx \mathrm{RF}_{\mathbf{v_i},I(\mathbf{x})}(\mathbf{l})\,.$$

The approach can be optimized in terms of memory requirements and processing time by using a subset of $\mathcal{S}$ for clustering. Memory requirements are lowered since only the subset needs to be kept in memory. Run time is reduced since subset selection is fast compared to the reduction in time during clustering.

Clustering not only reduces the storage requirements but additionally shortens preprocessing times significantly. Fitting a single two lobe RF requires about 20 ms, leading to a total preprocessing time of about 32 hours for a $256^2$ BTF with $|M_i| = |M_r| = 81$. The long time is almost entirely due to the large number of non-linear lobes that need to be fit. Compared to this, fitting clustered non-linear RFs requires only about 5 hours if 256 ABRDF clusters are employed. Here, total processing time is dominated by the cost for determining optimal clusters, which is mainly caused by the high dimensional-

| | Bump | LAF | SLAF | RFNL | RFNLC | CMF | VPCA | LPCA |
|---|---|---|---|---|---|---|---|---|
| Aluminium | 8.4% | 8.3% | 8.0% | 5.8% | 5.9% | 12.9% | 2.9% | 2.4% |
| Corduroy | 21.6% | 16.3% | 13.1% | 9.4% | 9.6% | 9.6% | 4.7% | 5.2% |
| Proposte | 14.4% | 11.4% | 10.7% | 9.3% | 9.7% | 9.4% | 4.7% | 4.8% |
| Stone | 16.2% | 13.1% | 12.3% | 9.2% | 9.8% | 10.1% | 7.0% | 5.1% |
| Wallpaper | 6.7% | 6.9% | 6.6% | 5.1% | 5.6% | 6.1% | 3.4% | 3.1% |
| Knitted Wool | 19.0% | 13.0% | 10.2% | 8.3% | 8.6% | 8.4% | 4.1% | 4.9% |

**Tab. 3.2: RMS approximation error for various BTFs and compression schemes.**

ity of the ABRDF. Fortunately, these costs can be reduced to a total of about two hours by applying principal components analysis.

## 3.6 Following Compression Methods

The development of BTF compression based on RFs marked an important step in the development of real-time, high-quality BTF rendering methods since RFs enable high quality compression of materials with significant mesostructure, which was previously not possible. Compared to techniques published earlier than the RF technique, RFs achieve a very good trade-off between rendering speed, memory requirements and reconstruction quality. Due to the large interest in accurate material representations, several other techniques for compression and rendering of measured BTFs were published [371] afterwards. To assess the performance of the RF method in comparison with the novel techniques, many aspects should be considered (e.g., compression rate, reconstruction speed, memory requirements, preprocessing time, numerical stability, etc.).

Concerning the goal of BTF rendering, the two most important characteristics are reconstruction error and compression ratio. Table 3.2 compares the average approximation error for BTF representations allowing for real-time decompression: Bump mapping with average BRDF, two Lafortune lobes (LAF) [348] with additional scaling term (SLAF) [89], two RF lobes (RFNL) clustered to 1024 bins (RFNLC), chained matrix factorization with 512 clusters (CMF) [507], per-view factorization with 4 components (VPCA) [447], and local PCA with 8 components (LPCA). The table shows the RMS errors, which is suitable for absolute comparisons and thus predictive rendering, but it remains questionable whether this metric provides insights concerning the similarity as perceived by human observers. It can be seen that RFs achieve very good quality but are inferior to data driven approaches based on PCA.

Table 3.3 compares the memory requirements for the various representations. Clearly, clustered RFs result in very compact representations, which makes them very suitable for scenes containing large numbers of different materials. Aggregating these two com-

| | Raw | Bump | LAF | SLAF | RFNL | RFNLC | CMF | VPCA | LPCA |
|---|---|---|---|---|---|---|---|---|---|
| $128^2$ | 0.6 GB | 0.2 MB | 0.6 MB | 5.9 MB | 48 MB | 3.0 MB | 1.9 MB | 38 MB | 11.1 MB |
| $256^2$ | 2.4 GB | 0.8 MB | 2.1 MB | 23.5 MB | 193 MB | 3.1 MB | 1.9 MB | 152 MB | 11.9 MB |
| $512^2$ | 9.6 GB | 3.2 MB | 8.4 MB | 94.0 MB | 772 MB | 3.5 MB | 2.1 MB | 608 MB | 15.1 MB |

**Tab. 3.3: Storage requirements for various BTF sizes.**

parisons, RFs still represent a suitable representation if accurate, real-time rendering of scenes with several materials is targeted. Among the novel techniques, only the local PCA technique achieves better results. An advantage of the RF method compared to the local PCA technique is that use of a larger number of clusters leads to less visible loss of crispness, which was pointed out as the weak property of the local PCA approach by Vasilescu and Terzopoulos [532] already. More detailed comparisons of existing techniques can be found in [361, 457, 371].

## 3.7 Discussion

In this chapter a method for compression of BTFs was presented which features a very good trade-off between memory requirements, reconstruction quality, and reconstruction speed. While it belongs to the category of parametric BTF models, which are known to yield relatively large compression errors, its main advantage is the combination of parametric modeling and linear interpolation. This approach enables highly accurate modeling of subsets of the data, and efficient interpolation between these subsets. As a result, reconstruction errors are much smaller than in comparable parametric BTF models and reconstruction remains almost as fast. Due to clustering of fitted RFs, memory requirements are also comparable.

Compared to the most recently developed approaches for BTF compression, the RF technique is still highly compatible in terms of the trade-off of reconstruction quality, memory requirements, and reconstruction speed. It is therefore very suitable for archiving purposes and for real-time use, which is essential for implementation of rendering algorithms. A detailed description of algorithms implementing real-time rendering from RF compressed BTFs can be found in Chapter 9.

# Chapter 4

# Synthesis of Near-Regular Textures and BTFs

Sampled BTFs are currently the most accurate general-purpose representation for surface materials. Yet, as Chapter 2 showed, some problems prohibit direct use of measured BTFs for real-time predictive rendering. The first limitation, the huge memory requirements, can be eased significantly by applying compression techniques like the one presented in the previous chapter. Removing the second limitation, the mismatch between the size of the acquired material samples (typically at most several square centimeters) and the size of the surface the materials are applied to (commonly several hundred square meters), is the goal of this chapter.

A simple solution to the problem is repetition of textual elements, which is commonly called *texture tiling*. Unfortunately, in cases with non-regular texture, tiling leads to disturbing repetition artifacts (cf. Figure 4.1) which should be eliminated for predictive rendering applications. To solve this problem, texture synthesis methods were devised which are supposed to enlarge the texture sample while preserving its characteristic visual textual properties. While these methods are quite successful for a large variety of textures, they often fail for near-regular materials, which are commonly employed in textile or car interior design.

Therefore, in the following section, an approach for faithful synthesis of these near-regular textures is described, which was first published in [387, 388]. Although the description of the approach is limited to synthesis of color textures, it can efficiently be extended to more accurate material representations like sampled BTFs. Therefore, in the second section in this chapter, two approaches for BTF synthesis will briefly be described.

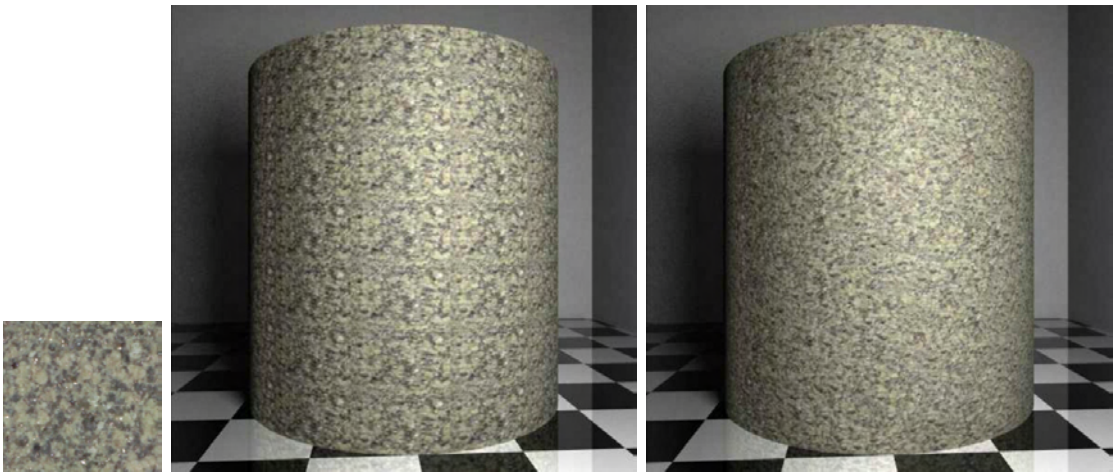


**Fig. 4.1: A stone BTF (left: frontal view) is once tiled on a cylinder (center) and once enlarged before texturing (right).**

# 4.1 Fractional Fourier Texture Masks

## 4.1.1 Problem Description

During the last years, the area of texture synthesis has been devoted significant work from researchers in computer graphics and computer vision. The developed algorithms achieve impressive synthesis results for various kinds of textures. Nevertheless, as shown in Chapter 2 not all kinds of textures can currently be handled in a pleasing manner. Especially near-regular textures (cf. Figure 2.15) that contain global, regular structures as well as stochastic deviations from regularity are still very difficult if not impossible to synthesize faithfully. Examples of this type of texture are frequently found in the real world and are typically highly relevant for Virtual Prototyping applications: Most textiles (e.g., used for clothing, furniture, or car interiors) and construction elements (walls, floors, grid structures, corrugated sheet roofs) fall into this category, and even natural objects like water waves, feathers and fur.

The specific difficulty of these textures originates from the strong mixture of regular patterns which govern global structure, and the subtle, yet characteristic deviation from this regularity. Since the regular patterns may be of arbitrary scale, existing sample-based synthesis algorithms fail to faithfully reproduce these textures due to restrictions of neighborhood sizes and lack of adequate distance functions.

In the following a technique for separation of regular structures from textures using fractional Fourier analysis is described. Due to their deterministic nature, new images containing these regular structures can be synthesized. These images, called fractional Fourier texture masks (FFTMs) in the following, can be used to guide sample-based

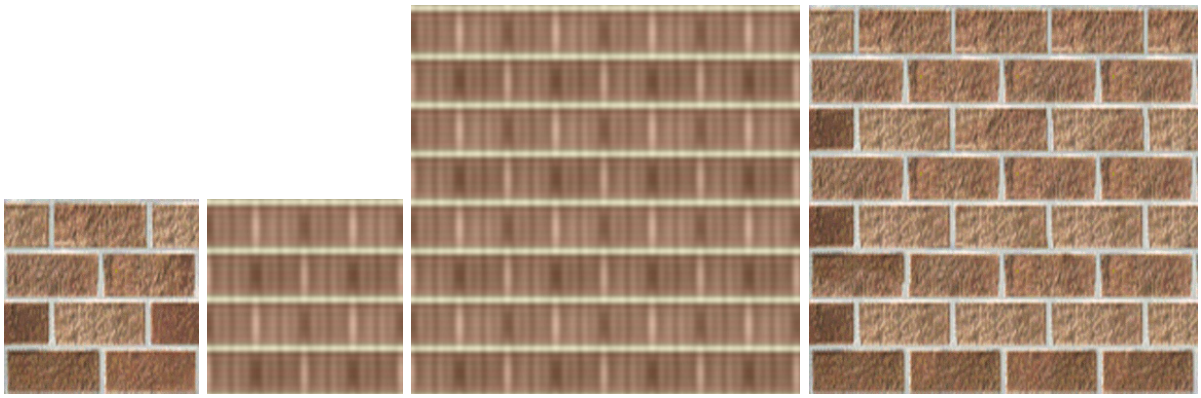


**Fig. 4.2: Texture synthesis using FFTMs. From left to right: input texture, extracted regular texture features, enlarged texture mask (FFTM), patch-based synthesis result including mask information featuring no broken features and correct regularity.**

synthesis algorithms in faithful selection and placement of copied pixels or patches, effectively enforcing global, regular structure and therefore leading to drastically improved synthesis quality for near-regular textures.

The method for synthesis of near-regular textures is composed of three major steps (cf. Figure 4.2):

1. Separation of the dominant regular structure from irregular texture detail using the fractional Fourier transform and an intensity filter,

2. Synthesis of the regular structure based on the inverse fractional Fourier transform,

3. Addition of irregular texture detail by extended sample-based texture synthesis.

In the following, first some background and implementation details of the algorithms for separation and synthesis of the regular texture structure are given. Then, it is shown how the resulting fractional Fourier texture masks can be used to guide existing sample-based texture algorithms to faithfully reproduce both local and global structure. The description of the technique ends with the presentation and discussion of results, after which some final conclusions are drawn.

### 4.1.2 Fractional DFT Analysis

In order to decompose textures into regular and irregular parts it is first necessary to define what exactly the terms *regular* and *irregular* refer to. An intuitive distinction between regular and irregular structures is that structures are perceived as regular if they

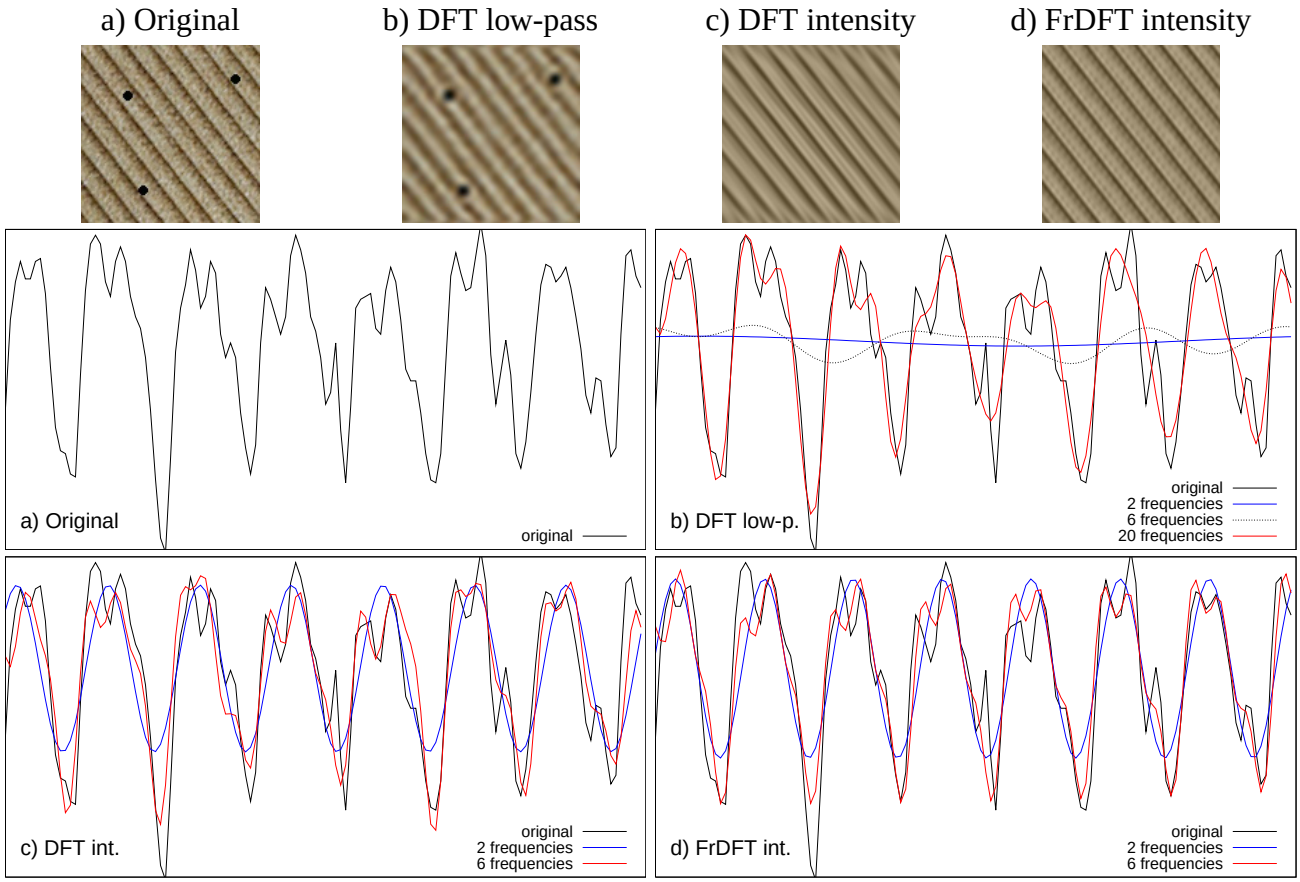



**Fig. 4.3: Results of applying filters to a corduroy texture sample with black dots added at random locations. Whereas the images show the texture sample, the curves visualize the luminance values of the pixels of the first scanline. While the lowpass filter result (b) still contains irregular structures and about 20 basis functions are required for a reasonable approximation of the 1D signal, the intensity filter (c) performs much better using just 6 frequencies. Nevertheless, the leakage error can only be eliminated applying the FrDFT intensity filter (d).**

occur (at least nearly) periodically while irregular structures are distributed in a different, more complex way. This informal definition suggests that representing a texture by periodic basis functions should enable efficient determination and extraction of regular structures.

Such a representation can be derived by applying a discrete Fourier transformation (DFT) which transforms a signal into a representation based on a sum of sine and cosine functions of varying frequencies. Given the resulting frequency spectrum, the question of which and how many frequencies are relevant for the regular part of the image signal arises. If too few frequencies are taken into account, most of the structure is smoothed out. If too many frequencies are selected, the signal itself is better reproduced but at the cost of introducing irregular structure.

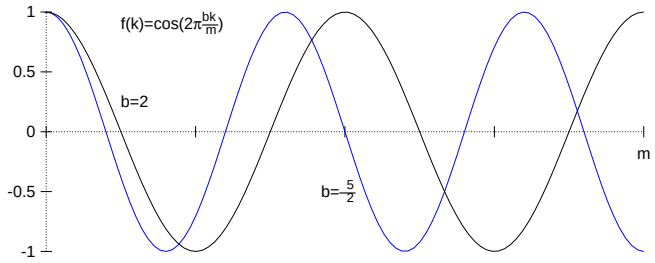



**Fig. 4.4: Fractional frequency. Representing the perfectly regular fractional frequency $b = \frac{5}{2}$ by integer frequencies requires a large amount of basis functions, since the corresponding signal is not periodic w.r.t. the range $[0, m]$.**

One way to select the relevant frequencies is to apply a low pass filter to the image signal. In previous approaches for texture synthesis this idea is incorporated by building image pyramids and synthesizing the low-resolution levels first. Unfortunately, in general these low frequencies do not correspond to the regular structures directly, which is shown in Figure 4.3b.

A more appropriate selection criterion is based on two observations. First, since the regular structures in the image signal dominate the overall appearance of most near-regular textures (see, e.g., the corduroy sample in Figure 4.3), they carry a much larger amount of energy than the irregular patterns. Second, while the energy contained in strongly visible irregular structures is distributed among a large number of frequencies, the main part of the energy contained in the periodic structures is carried by a few frequencies already (compare Figure 4.3c). Therefore, regular parts of the texture can be determined by selecting the frequencies carrying the most energy. In the following, this selection criterion is called an *intensity* filter.

While application of the intensity filter to the DFT of the signal yields good results already, the leakage effect of the DFT introduces artifacts along the borders of the image as illustrated in Figure 4.3c. This prohibits high quality synthesis of the extracted regular structure since the erroneous regions would be contained repeatedly in the synthesized image. The reason for the existence of leakage problems is shown in Figure 4.4, where both functions represent perfectly periodic signals restricted to the interval $[0, m]$. With respect to the given interval, only the function with $b = 2$ is periodic, the one with $b = \frac{5}{2}$ is not. Applying the intensity filter to the DFT always yields a signal periodic w.r.t. the given interval, which prohibits faithful representation of the perfectly regular frequency $b = \frac{5}{2}$. For a texture the restriction of the signal to such a window is usually defined implicitly when capturing the texture by a photograph. The window is typically neither perfectly aligned with the regular structure nor does it contain an *integer* amount of oscillations of the regular structure. A solution for this problem is the use of a fractional discrete Fourier transform.

**Definition (Fractional Frequency)** For a function $f(k) = \cos(2\pi\frac{b}{m}k)$ defined on $[0, m]$, $b \in [0, \frac{m}{2}]$ is called the frequency with respect to $m$. $b$ is called a *fractional frequency* if $b \notin \mathbb{Z}$. For a two-dimensional function

$$f(k_x, k_y) = \cos\left(2\pi\left(\frac{b_x k_x}{m_x} + \frac{b_y k_y}{m_y}\right)\right)$$

the analogon holds for a frequency pair $(b_x, b_y)$ with respect to $m_x$ and $m_y$.

Synthesizing the regular structure by continued evaluation of periodic functions requires exact knowledge of the fractional frequencies corresponding to the regular structures. A method to analyze these frequencies is the fractional Fourier transform (FrDFT, also called linear FrDFT [576]) [14]. While the DFT of a sequence $a_j, j \in [0, m[$ is a function $FT : \mathbb{Z} \to \mathbb{C}$ with

$$FT(z) = \frac{1}{m}\sum_{j=0}^{m-1} a_j \mathrm{e}^{-2\pi i\frac{zj}{m}},$$

the fractional Fourier transform of a sequence $a_j, j \in [0, m[$ is a function $FT : \mathbb{R} \to \mathbb{C}$ with

$$FT(k) = \frac{1}{m}\sum_{j=0}^{m-1} a_j \mathrm{e}^{-2\pi i\frac{kj}{m}}$$

or for a two-dimensional texture

$$FT(k_x, k_y) = \frac{1}{m_x m_y}\sum_{j_x=0}^{m_x-1}\sum_{j_y=0}^{m_y-1} a_{j_x j_y} \mathrm{e}^{-2\pi i\left(\frac{k_x j_x}{m_x}+\frac{k_y j_y}{m_y}\right)}.$$

### 4.1.3 Extraction of Regular Structures

The following property provides the key idea to extract the exact fractional frequency of a signal: The absolute value $|FT(k)|$ of the FrDFT of a *complex* periodic sequence of the form $a_j = F_b e^{2\pi i\frac{bj}{m}}, j \in [0, m[, F_b \in \mathbb{C}$ with frequency $b \in \mathbb{R}$ reaches its maximum for $k = b$ [14] and $FT(b) = F_b$ is the Fourier coefficient that determines intensity and phase (see Figure 4.5). To exploit this property (which analogously holds for frequency pairs) in the context of texture synthesis, two problems have to be solved. First, textures are real-valued. Second, the regular structure of most relevant textures is contained in more than one frequency and therefore not only a single frequency has to be determined.

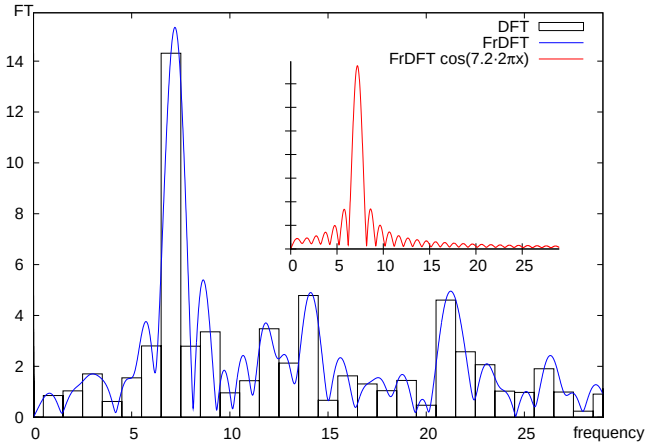



**Fig. 4.5: Comparison of absolute DFT and FrDFT coefficients for the curve in Figure 4.3 after subtraction of mean value. The dominant regular structure is caused by frequency 7.2. The small plot shows the FrDFT of the function** $\cos(7.2 \cdot 2\pi x)$**.**

A widely accepted approximation to the first problem is the computation of the *analytic signal* from the real-valued input by computing the Hilbert transform [45]. This first requires computation of the DFT coefficients $F_{b_x,b_y}$ of the input texture. Then, assuming the texture has even width $m_x$ and even height $m_y$, new DFT coefficients $\tilde{F}_{b_x,b_y} = w(x,y)F_{b_x,b_y}$ are computed, with

$$w(x,y) = \begin{cases} 1 & (x,y) \in \left\{(0,0), (\frac{m_x}{2},0), (0,\frac{m_y}{2}), (\frac{m_x}{2},\frac{m_y}{2})\right\} \\ 2 & (x,y) \in \left[1,\frac{m_x}{2}-1\right] \times [0,m_y-1] \cup \left[0,\frac{m_x}{2}\right] \times \left[1,\frac{m_y}{2}\right] \\ 0 & else \end{cases}$$

For odd widths or heights formulas are similar. Afterwards, inverse DFT is applied to $\tilde{F}_{b_x,b_y}$, resulting in the desired, complex-valued sequence.

Solving the second problem is more difficult. If the regular structure is contained in several frequency pairs (as usual), the FrDFT is the sum of the FrDFT of each of the pairs. As the small plot in Figure 4.5 shows, $|FT(k)|$ contains many local maxima even for a function determined by a single dominant fractional frequency. The FrDFT of a real sequence as in the large plot of Figure 4.5 therefore contains various local maxima of which some correspond to fractional frequencies contained in the regular structure and some do not. Therefore, in the following a method for identifying the correct maxima in $FT(k_x,k_y)$ corresponding to the fractional frequencies that carry the regular structure is described.

The method is based on the following theorem:

**Theorem** Let $d := 2\pi(k-b)$. Then for $k, b \in [0, m[$

$$|FT_b(k)|^2 \approx \mathrm{sinc}^2(k-b)$$

with

$$\mathrm{sinc}(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} & x \neq 0 \\ 1 & x = 0 \end{cases}.$$

**Proof** The Taylor series of $1 - \cos\left(\frac{d}{m}\right)$ gives

$$1 - \cos\left(\frac{d}{m}\right) \approx \frac{d^2}{2m^2}$$

Therefore

$$\begin{aligned} |FT_b(k)|^2 &= \frac{1}{m^2}\left|\frac{1-e^{id}}{1-e^{\frac{id}{m}}}\right|^2 = \frac{1-\cos(d)}{m^2\left(1-\cos\left(\frac{d}{m}\right)\right)} \text{[14]} \\ &\approx \frac{2}{d^2}\left(1-\cos(d)\right) = \frac{2}{d^2}\left(1-\left(\cos^2\left(\frac{d}{2}\right)-\sin^2\left(\frac{d}{2}\right)\right)\right) \\ &= \frac{2}{d^2}\left(2\sin^2\left(\frac{d}{2}\right)\right) = \left(\frac{\sin\left(\frac{d}{2}\right)}{\frac{d}{2}}\right)^2 = \mathrm{sinc}^2(k-b). \end{aligned}$$

If frequency $b$ is described by Fourier coefficient $F_b$, then

$$|FT_b(k)|^2 \approx \left(|F_b|\,\mathrm{sinc}\,(k-b)\right)^2.$$

For the 2D case, following a similar argumentation,

$$|FT_{b_x,b_y}(k_x,k_y)|^2 \approx |F_{b_x,b_y}|^2 \cdot \mathrm{sinc}^2(k_x-b_x) \cdot \mathrm{sinc}^2(k_y-b_y). \qquad (4.1)$$

for $b_x, k_x \in [0, m_x[$ and $b_y, k_y \in [0, m_y[$.

Equation 4.1 shows that the magnitude $m_{l_i}$ of a local maximum at $k_{l_i} = (k_{x_i}, k_{y_i})$ is much smaller than the magnitude $m_g$ of the global maximum at $k_g = (k_{x_g}, k_{y_g})$. The relationship between $m_{l_i}$ and $m_g$ is approximately as follows (for $k_{x_i} \neq k_{x_g}$ and $k_{y_i} \neq k_{y_g}$):

$$\frac{m_g}{m_{l_i}} \approx \pi^2 \left|\left(k_{x_g}-k_{x_i}\right)\left(k_{y_g}-k_{y_i}\right)\right|.$$

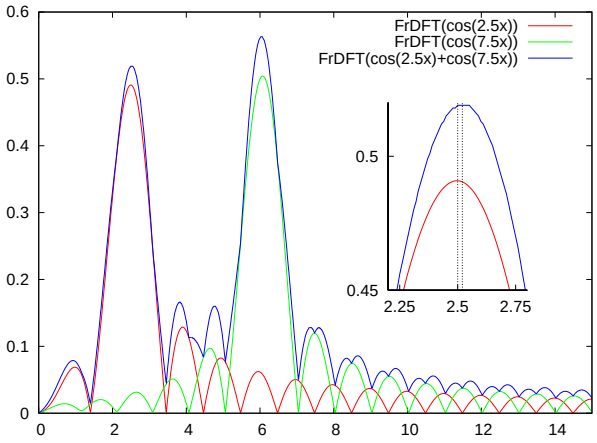



**Fig. 4.6: Superposition of two frequencies typically retains the FrDFT maxima. Enlarged view: the exact location of maxima may be shifted slightly.**

## Proof

$$\begin{aligned}\frac{m_g}{m_{l_i}} &= \frac{\left|FT_{k_{x_g},k_{y_g}}\left(k_{x_g},k_{y_g}\right)\right|}{\left|FT_{k_{x_g},k_{y_g}}\left(k_{x_i},k_{y_i}\right)\right|} \approx \frac{\left|F_{k_{x_g},k_{y_g}}\right|\cdot\left|\operatorname{sinc}\left(k_{x_g}-k_{x_g}\right)\cdot\operatorname{sinc}\left(k_{y_g}-k_{y_g}\right)\right|}{\left|F_{k_{x_g},k_{y_g}}\right|\cdot\left|\operatorname{sinc}\left(k_{x_i}-k_{x_g}\right)\cdot\operatorname{sinc}\left(k_{y_i}-k_{y_g}\right)\right|} \\ &= \frac{1}{\left|\frac{\sin\left(\pi\left(k_{x_i}-k_{x_g}\right)\right)}{\pi\left(k_{x_i}-k_{x_g}\right)}\cdot\frac{\sin\left(\pi\left(k_{y_i}-k_{y_g}\right)\right)}{\pi\left(k_{y_i}-k_{y_g}\right)}\right|} = \pi^2\left|\left(k_{x_g}-k_{x_i}\right)\left(k_{y_g}-k_{y_i}\right)\right| \qquad (4.2)\end{aligned}$$

since the distance $|k_g - k_{l_i}|$ between global and local minima of the sinc function – and thus approximately the FrDFT – equals $z + \frac{1}{2}$ in each dimension ($z \in \mathbb{Z}$).

Due to this strong difference between the magnitudes of local and global maxima, the global maximum should be well preserved even if multiple fractional frequencies are contained in the sequence (cf. Figure 4.6). In addition, the most intense maximum should remain the global maximum, which turned out to be a valid assumption during all tests.

The basic method for extraction of relevant frequencies is therefore the following: First, the frequency pair with largest absolute value of the corresponding fractional Fourier coefficient is determined. Then its contribution to the analyzed image is removed and a FrDFT transformation is performed on the remaining image. This process is repeated until the absolute value of the largest remaining Fourier coefficient is below some threshold (cf. Figure 4.7). In each step largest values are determined by first sampling the coefficients with a fixed step size and then applying a standard maximization algorithm, starting at the sample with largest value.

Unfortunately, extracting frequencies in this way is very time consuming (the runtime of the algorithm is roughly $O(m_x m_y \frac{m_x}{d_x}\frac{m_y}{d_y} n)$, where $d_x$ and $d_y$ denote the distance

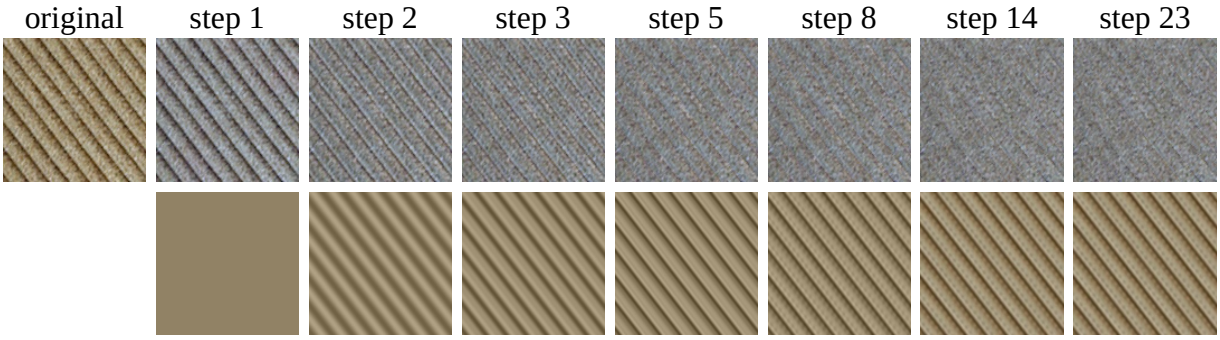



**Fig. 4.7: Steps of the Fourier analysis algorithm. Top: remaining texture elements, bottom: extracted elements.**

| Threshold | 0.5 | 1 | 2 | 5 |
|---|---|---|---|---|
| Frequency pairs found | 175 | 76 | 35 | 14 |
| Runtime | 97 s | 48 s | 27 s | 11 s |

**Tab. 4.1: Runtime and result size of the FrDFT analysis for various thresholds and the texture from Figure 4.3 measured on a 2.4 GHz PC.**

in $x$ and $y$ direction between values for which the FrDFT coefficient is computed and $n$ denotes the number of extracted frequencies). The run-time complexity of the algorithm can be improved in two ways. First, the maximum search in the 2D-FrDFT can be reduced to the 1D case, because for fixed $k_x$ or fixed $k_y$ Equation 4.1 is a 1D $\mathrm{sinc}^2$ function. To determine a 2D maximum, it is therefore convenient to first search a 1D maximum with fixed $k_y$ and then use its position as fixed $k_x$.

Applying this modification, the run-time complexity reduces to $O(m_x m_y(\frac{m_x}{d_x}+\frac{m_y}{d_y})n)$ but still remains high. Therefore, a second optimization restricts the regions searched for maxima. As visible in Figure 4.5, the position of a FrDFT global maximum can roughly be determined using the DFT. One is only looking for the most intense frequencies and thus only the highest maxima of $FT$. The DFT yields the values of $FT$ for $k_x, k_y \in \mathbb{Z}$ and therefore one only needs to search the $[-1,1]^2$ regions around DFT coefficients that are larger than a threshold. Choosing a high threshold can save a large amount of time (cf. Table 4.1). Applying this second modification reduces the run-time to $O(m_x m_y(\frac{2}{d_x}+\frac{2}{d_y})c)$, where $c$ denotes the number of DFT values above the threshold.

### Optimization of Extracted Frequencies

The extracted frequency pairs are near-optimal only since interference effects caused by the summation of multiple frequencies shift the location of maxima slightly (cf. Figure 4.6). To compensate for this effect, the intensity filter based on Fourier analysis

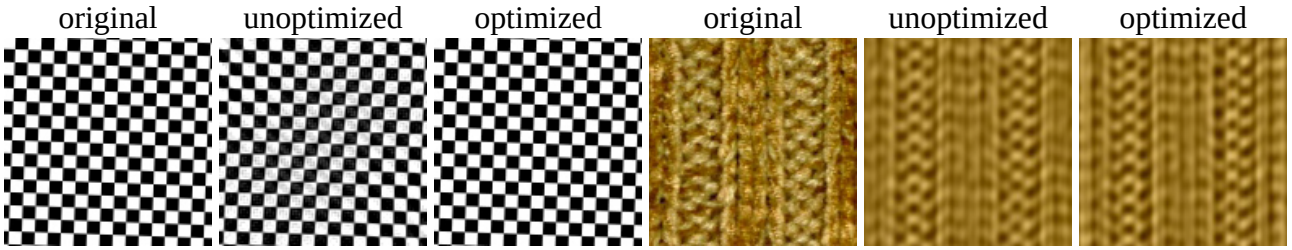



**Fig. 4.8: Effects of optimization of extracted frequencies for two textures.**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| cracker | Frequency pairs | 7 | 13 | 16 | 23 | 29 | 44 |
| | Improvement | 1.0% | 1.0% | 0.8% | 1.6% | 1.5% | 1.5% |
| | Runtime | 4 s | 140 s | 120 s | 25 min | 41 min | 115 min |
| knitted wool | Frequency pairs | 6 | 9 | 14 | 20 | 25 | 36 |
| | Improvement | 4.7% | 3.9% | 5.1% | 6.3% | 6.7% | 7.2% |
| | Runtime | 13 s | 120 s | 430 s | 42 min | 90 min | 140 min |

**Tab. 4.2: Runtime of the optimization procedure depending on the maximum number of extracted frequency pairs per color channels.**

is rewritten as an energy minimization problem. Since the intensity filter extracts the frequencies with the most energy, it minimizes the least squares color difference

$$E = \frac{1}{m_x \cdot m_y} \sqrt{\sum_{x=0}^{m_x-1} \sum_{y=0}^{m_y-1} \Big( T_{in}(x,y) - M_{in}(x,y) \Big)^2} \tag{4.3}$$

between the input texture $T_{in}$ and $M_{in}$, a texture with equal size containing the regular structure due to the extracted frequency pairs which is generated from inverse FrDFT transform

$$M_{in}(x,y) = \sum_{k=1}^{n} F_k e^{2\pi i \left( \frac{x b_{x_k}}{m_x} + \frac{y b_{y_k}}{m_y} \right)}. \tag{4.4}$$

of the $n$ frequency pairs $b_k = (b_{x_k}, b_{y_k})(k \in [1,n])$ and corresponding Fourier coefficients $F_k$. Obviously, minimization of this energy function will result in frequency pairs that optimally represent the input texture. In the implemented example, this minimization is achieved by applying non-linear optimization to the frequency pairs extracted by fractional Fourier analysis using the Levenberg-Marquardt [422] algorithm, which turned out to give better results than comparable approaches like Downhill-Simplex [422].

As Table 4.2 shows, the run-time of the optimization procedure mainly depends on the maximum number of extracted frequencies per color channel. The approximately cubic dependence makes the approach useful for cases with limited dimensionality only.

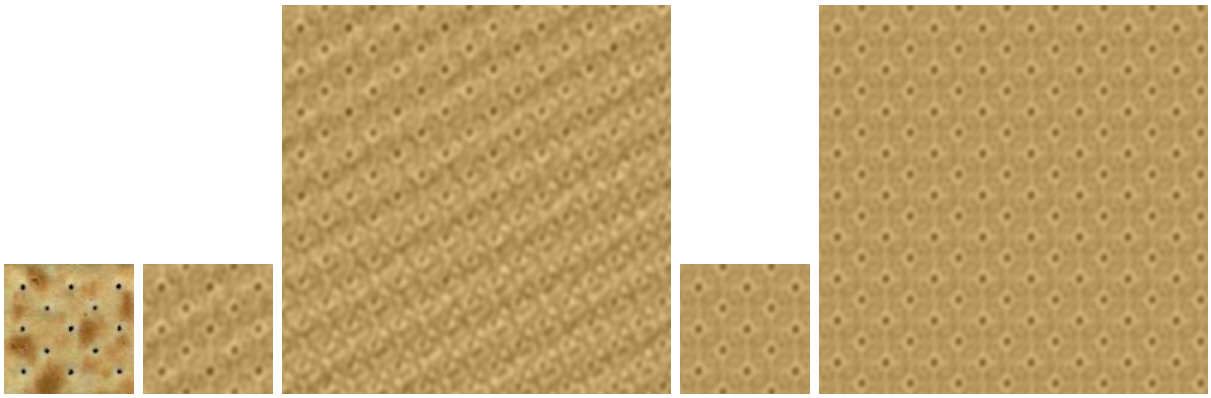


**Fig. 4.9: Effects of enforcement of the base frequency pair for the cracker texture. From left to right: original, extracted regular structure, enlarged regular structure showing undesired interference effects, enforcement of base frequency pair, and correct, enlarged structure.**

Significant additional influence can be contributed to the size of the analyzed texture. Fortunately, as the improvements suggest, the unoptimized extracted frequencies are usually very accurate already (cf. Figure 4.8). Therefore, the optimization procedure can often safely be skipped.

### Enforcement of Base Frequency Pair

Optimizing the extracted frequencies following the above scheme leads to very accurate regular structures. Unfortunately, optimizing accuracy is insufficient in some cases. Figure 4.9 shows such an example. The cracker texture contains regularly placed, small, black dots. In order to represent these small features including their sharp borders, mixtures of several frequency pairs at distinct ratios are required. Typically, sharp features are generated by a base frequency pair and multiples of it. If the extracted frequency pairs deviate from these distinct ratios (e.g., caused by numerical inaccuracies, by local minima the non-linear optimization gets stuck in, or by influences from irregular texture parts), undesired interference artifacts become visible outside the area of the original texture sample. Therefore, regularity is clearly not captured in these regions. This effect is shown in Figure 4.9 and the corresponding Table 4.3: The black dots are generated by a base frequency pair of $(b_{x_b}, b_{y_b}) = (2.6782, 3.1042)$. Before enforcing it, extracted frequencies (slightly) deviate from multiples of the base pair, resulting in undesired interference effects. Afterwards, the distinct ratios are enforced perfectly, eliminating any undesired interference.

The energy minimization scheme described above is not capable of enforcing distinct ratios between extracted frequencies but this requirement can be introduced based on the following fact: For all (mixtures of) regular texture structures there exists an image

| k | ch. | $b_{x_k}$ | $b_{y_k}$ | $Re$ | $Im$ | ch. | $b_{x_k}$ | $b_{y_k}$ | $Re$ | $Im$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Y | 0.0 | 0.0 | 156.8 | 0.0 | Y | 0.0 | 0.0 | 156.8 | 0.0 |
| 2 | $C_r$ | 0.0 | 0.0 | -34.1 | 0.0 | $C_r$ | 0.0 | 0.0 | -34.1 | 0.0 |
| 3 | $C_b$ | 0.0 | 0.0 | 19.8 | 0.0 | $C_b$ | 0.0 | 0.0 | 19.8 | 0.0 |
| 4 | Y | 1.1103 | 2.5878 | -2.8 | -7.8 | Y | 119.9654 | 3.1042 | 3.9 | 2.3 |
| 5 | Y | 119.9744 | 3.1673 | 4.5 | 1.2 | Y | 2.6782 | 9.3126 | -4.0 | 0.1 |
| 6 | Y | 2.5690 | 4.5923 | -4.2 | 1.5 | Y | 5.3564 | 6.2084 | -2.4 | -3.2 |
| 7 | Y | 5.5241 | 6.3059 | -4.2 | -0.8 | Y | 10.7128 | 6.2084 | 3.2 | -1.8 |
| 8 | Y | 2.7014 | 9.3199 | -4.1 | 0.3 | Y | 5.3564 | 121.7916 | 2.4 | -2.8 |
| 9 | Y | 5.1943 | 121.5655 | 3.3 | 1.8 | Y | 2.6782 | 118.6872 | 0.1 | -3.6 |
| 10 | Y | 10.7947 | 6.1151 | 3.2 | -1.7 | Y | 117.2872 | 6.2084 | 2.7 | -2.2 |
| 11 | Y | 2.5475 | 118.5844 | 2.7 | -2.3 | Y | 8.0346 | 3.1043 | 1.3 | -2.9 |
| 12 | Y | 8.1016 | 3.0786 | 1.2 | -3.2 | Y | 8.0346 | 9.3126 | -0.1 | -3.1 |

**Tab. 4.3: Most significant extracted frequency pairs before (left) and after (right) enforcement of base frequency pair. The color channel (ch.), frequencies ($b_{x_k}, b_{y_k}$), and coefficients $F_k = Re + i \cdot Im$ are shown.**

size $(m_{x_t}, m_{y_t})$ such that continuing or cropping the structures leads to tileable textures, meaning that an integer amount of repetitions of the structure is contained in the texture (this fact was also used by Liu et al. [322]). In frequency space, $(m_{x_t}, m_{y_t})$ corresponds to a *base frequency pair* $b_b = (b_{x_b}, b_{y_b})$ such that all frequency pairs $b_k$ required for reconstruction of regular structures are integer multiples of $b_b$. Determining such a base frequency pair and enforcing frequency pairs for reconstruction to be integer multiples of this frequency leads to elimination of undesired interference.

Applying this observation leads to the following minimization problem: Find a base frequency pair $(b_{x_b}, b_{y_b})$, $n$ pairs of integers $(m_{x_k}, m_{y_k})$, and corresponding reconstruction weights $r_k$ such that

$$E = \frac{1}{m_x \cdot m_y} \sqrt{\sum_{x=0}^{m_x-1} \sum_{y=0}^{m_y-1} \Big( T_{in}(x,y) - M_{in}(x,y) \Big)^2} \tag{4.5}$$

with

$$M_{in}(x,y) = \sum_{k=1}^{n} r_k e^{2\pi i \left( \frac{x m_{x_k} b_{x_b}}{m_y} + \frac{y m_{y_k} b_{y_b}}{m_x} \right)} \tag{4.6}$$

is minimized. Please note that the reconstruction weights $r_k$ need not necessarily be the Fourier coefficients. It is, e.g., possible to assume a fixed domain of texture values (typically $[0, 255]$ for standard textures) which allows for introduction of an additional clamp function into the above equation.

Unfortunately, the energy in Equation 4.5 can trivially be minimized by setting the

base frequency pair components to arbitrarily small values, such that the adjusted frequency pairs converge to the ones extracted with FrDFT analysis. To avoid this, assumptions about the base frequency pair have to be made. One assumption typically fulfilled is that both components of the base pair are greater than two, i.e., that the input texture contains at least two full periods of texture in each direction (e.g., the texture in Figure 4.9 features about three periods in each direction).

The above optimization problem is a non-linear, mixed integer, constrained optimization problem with a non-convex, differentiable target function. Such problems are known to be difficult to solve, and often require inefficient algorithms to determine an optimal solution despite continuous advances in the field [51]. Since the number of variables is quite high (20 pairs of integers and 20 weights are quite common) an inefficient algorithm is not acceptable.

It is therefore more suitable to apply a combination of relaxation and selection steps: The relaxation step determines an optimal solution without the integer constraints, the selection step searches an optimal solution in the vicinity of the solution found by the relaxation step including the integer constraints. This method follows a standard approach from non-linear mixed integer optimization by combining non-linear and integer optimization methods. The approach provides no guarantees on determining a global optimum and therefore the quality of the solution heavily depends on a good initial guess. Fortunately, such an initial solution can efficiently be computed from Equation 4.3 by applying FrDFT analysis as described above.

In the following, the reconstruction weights $r_k$ equal the Fourier coefficients $F_k$, which allows for an especially efficient implementation, since the relaxation step reduces to FrDFT analysis and since no iterations between relaxation and selection steps are necessary. The selection step needs to minimize the reconstruction error while simultaneously enforcing a base frequency pair. Unfortunately, direct optimization of this error function is time consuming due to integer restrictions.

Therefore, a different multi-step selection method is employed. In a first step, several base frequency candidate pairs $(b_{x_j}, b_{y_j})$ closely fitting the respective components of the extracted pairs $b_i = (b_{i_x}, b_{i_y})$, i.e., locally minimizing

$$\varepsilon_x = \frac{\sum_{i=1}^{k} |F_i| \cdot \left(\min_{n\in\mathbb{N}} \left\{ \left\| n \cdot b_{x_j} - b_{i_x} \right\|_1 \right\}\right)^2}{b_{x_j}^2 \sum_{i=1}^{k} |F_i|} \tag{4.7}$$

respectively

$$\varepsilon_y = \frac{\sum_{i=1}^{k} |F_i| \cdot \left(\min_{n\in\mathbb{N}} \left\{ \left\| n \cdot b_{y_j} - b_{i_y} \right\|_1 \right\}\right)^2}{b_{y_j}^2 \sum_{i=1}^{k} |F_i|} \tag{4.8}$$

are determined. Note, that the $x$ and $y$ components can be handled independently of each other. Due to the above assumption and since the base frequencies should not be much larger than the smallest extracted frequency pair components, $2 \leq b_{x_j} \leq \min\{b_{i_x}\}_{i\in[1,k]} + \delta_x$ and $2 \leq b_{y_j} \leq \min\left\{b_{i_y}\right\}_{i\in[1,k]} + \delta_y$ ($\delta_x$ and $\delta_y$ are application-dependent parameters, set to 4 in the conducted experiments). Such frequencies can efficiently be determined by uniformly sampling the range of possible frequencies and selecting samples corresponding to local minima. Determined minima can be improved by iteratively refining the sampling density in their vicinity.

Once the sets $\{b_{x_j}\}$ and $\{b_{y_j}\}$ of locally optimal frequencies are determined, the set $\{b_{x_j}\} \times \{b_{y_j}\}$ of possible base frequency pairs is computed and elements are sorted based on the combined approximation error $\varepsilon_x + \varepsilon_y$. Then, among the first $n$ of these, the pair resulting in least reconstruction error according to Equation 4.5 is selected as the base frequency pair. Finally, appropriate integer pairs and reconstruction weights for the extracted frequency pairs are computed.

The two-step selection operation requires some seconds on a standard PC for most input textures (e.g., 15 seconds for the example in Figure 4.9). The exact run-time depends on various parameters like the size and complexity of the input texture, the number of extracted frequencies, and the number of candidate pairs for which the reconstruction error is evaluated. For most textures the approach succeeds in removing undesired interference artifacts (cf. Figure 4.16 for another example). It nevertheless has problems with textures that violate the assumption that at least two periods need to be present in each dimension. This can be improved manually by making the user select the approximate size of a regular patch and optimizing the user input by searching the vicinity. Another problem of the algorithm is that it assumes a single structure in the texture. If independent structures exist (cf. Figure 4.18) either the input texture needs to be enlarged or the algorithm for extraction of base frequency pairs needs to be combined with a clustering approach that separates frequencies from each of the independent structures. Unfortunately, the notion of independent structures is very difficult to capture mathematically. Thus, such an approach might require user-intervention for generating meaningful results.

### Fractional Fourier Texture Masks

The analysis process outlined above determines frequency pairs $b_1, \ldots, b_n$ with respective Fourier coefficients $F_1, \ldots, F_n$. These frequencies can be synthesized using the

inverse DFT formula:

$$a_{j_x j_y} = \sum_{h=1}^{n} F_h e^{2\pi i \left( \frac{b_{h_x} j_x}{m_x} + \frac{b_{h_y} j_y}{m_y} \right)}.$$

If one chooses $j_x, j_y \in \mathbb{Z}$, the size of the synthesized texture is unlimited and the frequencies of the regular structures represent the parameters of a procedural texture containing these structures. This process is known as Fourier synthesis [569] and the output is called *fractional Fourier texture masks* (FFTMs) in the following. Examples of FFTMs are shown in the third row of Figure 4.12.

The frequency information can be used to calculate the size of a texture with *tileable* regular structures. A regular structure is tileable if the frequencies describing the structure are not fractional w.r.t. the texture size. In most cases, this is the base frequency pair $b_b$ determined from Equation 4.5. Thus

$$m'_x = m_x \cdot \frac{n_x}{b_{b_x}} \text{ and } m'_y = m_y \cdot \frac{n_y}{b_{b_y}} \tag{4.9}$$

produces tileable sizes for all $(n_x, n_y) \in \mathbb{N} \times \mathbb{N}$ and $n_x = n_y = 1$ is a single tile of the pattern as described by Liu et al. [323]. In the most general case, the texture contains multiple base frequency pairs $b_{b_i}$ with $i \in [0, n]$. In this case, the smallest common denominator $b_{b_c}$ of the $b_{b_i}$ has to be determined and used instead of $b_b$ in Equations 4.9.

With the notable exception of enforcing base frequency pairs, FrDFT analysis and synthesis are performed for each color channel separately. Various tests determined that transforming the RGB color data to $\mathrm{YC_rC_b}$ color space gives best results since this reduces deviation of the per-channel frequencies. In addition, for many textures the analysis turns out to become faster, because usually the Y-channel contains the most information.

### 4.1.4 Texture Synthesis

As described above already, existing texture synthesis algorithms have difficulties with reproducing near-regular textures since they need to reproduce both regular and irregular structures. Typically, existing techniques are more successful at handling stochastic textures with irregular structures. Therefore, combining these algorithms with FFTMs, which capture the regular elements of a texture, should lead to better synthesis methods that accurately handle textures containing both regular and irregular elements.

Currently, the most prominent techniques for texture synthesis are methods based on intelligent sampling. In the following, it is shown how FFTMs can be used with these

techniques (either pixel- or patch-based). The texture synthesized by such a method will be denoted by $T_{out}$, an FFTM with size equal to the desired output size by $M_{out}$, and the FFTM corresponding to input texture $T_{in}$ by $M_{in}$. In this way for every pixel in $T_{in}$ there is exactly one corresponding pixel in $M_{in}$, and the same holds for pixels in $T_{out}$ and $M_{out}$.

### Pixel-Based Synthesis

In the pixel-based synthesis approach, an output texture $T_{out}$ is generated from an input texture $T_{in}$ by selectively copying pixels. A pixel is chosen by comparing the neighborhood of the current pixel in $T_{out}$ to a candidate list of neighborhoods of pixels from $T_{in}$. The comparison of neighborhoods is typically carried out by computing the $L_2$ norm of the difference between neighborhood vectors $\vec{N}(T_{in}, x_{in})$ and $\vec{N}(T_{out}, x_{out})$. A typical neighborhood is shown in Figure 4.10.

Incorporating FFTMs into pixel-based synthesis algorithms can be achieved in different ways, of which the following two ones were implemented:

**Full neighborhood comparison.** This approach is inspired by the texture mask approach as proposed by Zhang et al. [611]. The neighborhood used to determine best matching pixels is extended from $N_{upper}$ to $N_{full} = N_{upper} \cup N_{lower}$. Whereas the similarity of pixels in $N_{upper}$ is measured by the standard $L_2$ norm

$$d_{TT} = \left\| \vec{N}(T_{in}, x_{in}) - \vec{N}(T_{out}, x_{out}) \right\|_2 , \tag{4.10}$$

for pixels in $N_{lower}$ the $L_2$ norm

$$d_{MM} = \left\| \vec{N}(M_{in}, x_{in}) - \vec{N}(M_{out}, x_{out}) \right\|_2 \tag{4.11}$$

of the respective FFTM values is evaluated. The contributions from both parts are summed into a single value after scaling the FFTM-similarities by a mask weight $c$. Please note that incorporating $d_{MM}$ can be interpreted as the previously missing way to improve the standard similarity measure to consider structural information as well.

**Cluster ID match.** The pixels of $M_{in}$ are clustered based on similar neighborhoods $N_{full}$ using k-means clustering. Whenever a pixel $x_{out}$ is synthesized, the cluster ID of the corresponding pixel is determined. This cluster ID defines a candidate set of pixels in $M_{in}$ and thus corresponding pixels $T_{in}$ from which the pixel to be copied is selected using the measure $d_{TT}$ defined above.

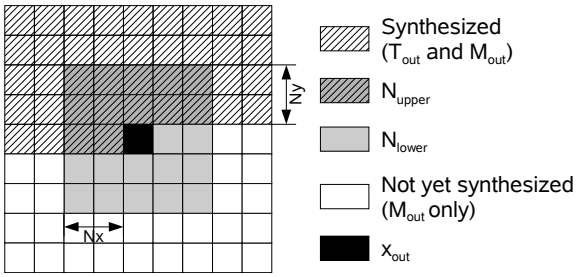



**Fig. 4.10: The neighborhoods $N_{upper}$ and $N_{lower}$ with size $N_x \times N_y$. $N_{upper}$ is defined for all textures and masks, but $N_{lower}$ is undefined for $T_{out}$.**

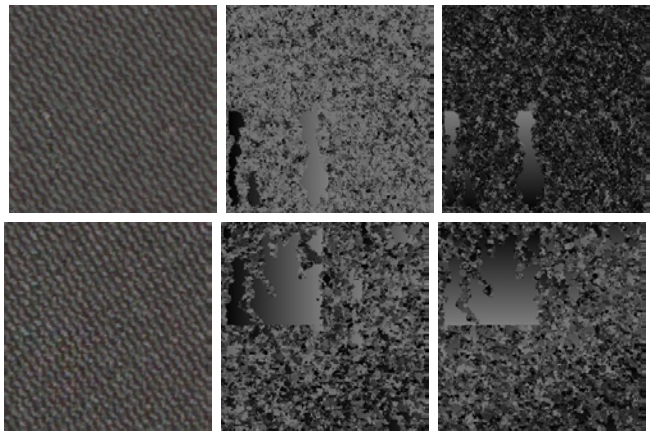


**Fig. 4.11: Synthesis results for the texture in Figure 4.12. Upper row: with k-coherence and cluster ID match. Results with full search and full neighborhood comparison are similar. Lower row: ANN search and full neighborhood comparison. Both examples used a $3^2$ neighborhood. The images on the right visualize $x$ and $y$ components of source coordinates respectively.**

To accelerate the actual neighborhood selection standard strategies like the k-coherence search of Tong et al. [520] and approximate nearest neighbor (ANN) search [311, 609] can be applied in both approaches. Experiments showed that best results are achieved for the combination of full neighborhood comparison with ANN search. An examination of the output source coordinates as illustrated in Figure 4.11 shows the reason for the better quality of the ANN search results. The apparent noise in the output texture is caused by very little spatial coherence in the source coordinates. The noise is *not* caused by problems in the k-coherence or the ANN search algorithm, because it occurs even with an exhaustive search that always finds the best matching neighborhood. If the ANN search with the full neighborhood search is used, the source coordinates show that small spatially coherent regions are copied from the input texture. This is due to the implementation of the ANN algorithm used [367], which favors neighborhood vectors close to the query vector if the tree was built with the vectors in scanline order. This observation is in accordance with the results of Ashikhmin [10].

Experiments determined that for reproducing simple structures with the ANN approach, a small $2^2$ neighborhood and a weight $c \approx 1$ produce good results. Reproduction of more complex structures like knitted wool requires a larger $6^2$ neighborhood to obtain good results and possibly a higher value for $c$. Usually, if $c < 0.1$ the mask has no effect, and if $c > 5$ the mask weight is too large. In case large neighborhoods are required, principal component analysis can be applied to reduce the dimension of the neighborhood vectors to about 16, which increases synthesis speed significantly while decreasing synthesis quality just slightly.

The algorithm can easily be extended to create tileable textures. Equation 4.9 can be used to calculate an appropriate output size for the regular structures. If one wraps around the neighborhoods at the border and adds a second pass for the pixels where the opposite neighborhood was undefined during the first pass, the irregular structures produced by the pixel synthesis fit without visible edges.

#### Patch-Based Synthesis

Incorporation of FFTMs into patch-based synthesis algorithms basically follows the same idea as for pixel-based methods. Whereas the quality of pixel-based algorithms was improved by completing the full neighborhood with values from FFTMs, the same can be applied to parts of patches that do not overlap with already synthesized pixels.

Additionally, to support iterative improvement of synthesized textures using the Graphcut algorithm [288] the pixel representation is extended from standard 3D RGB vectors to 6D vectors that additionally contain the respective mask values, scaled by the FFTM weight $c$. This way, one can simply reuse the standard $L_2$ norm to compute $d_{MM} + c \cdot d_{TT}$, which helps to find well fitting patches that additionally preserve the regular structure. Obviously, as for pixel-based synthesis, tileable textures can be synthesized by computing appropriate output sizes and wrapping around neighborhoods at borders.

### 4.1.5 Results

The above algorithms were tested with various texture samples. The pixel-based algorithm is very easy to use since the only manual parameters that need to be selected are the intensity filter threshold, the neighborhood size, and the mask weight. For synthesizing textures with the help of FFTMs a $3^2$ neighborhood was used which is (much) smaller than the largest structures in the textures. When omitting FFTMs much larger neighborhoods were used (for the images in Figure 4.12, from left to right $6^2$, $5^2$, $8^2$, $10^2$, and $10^2$) which increases synthesis times drastically.

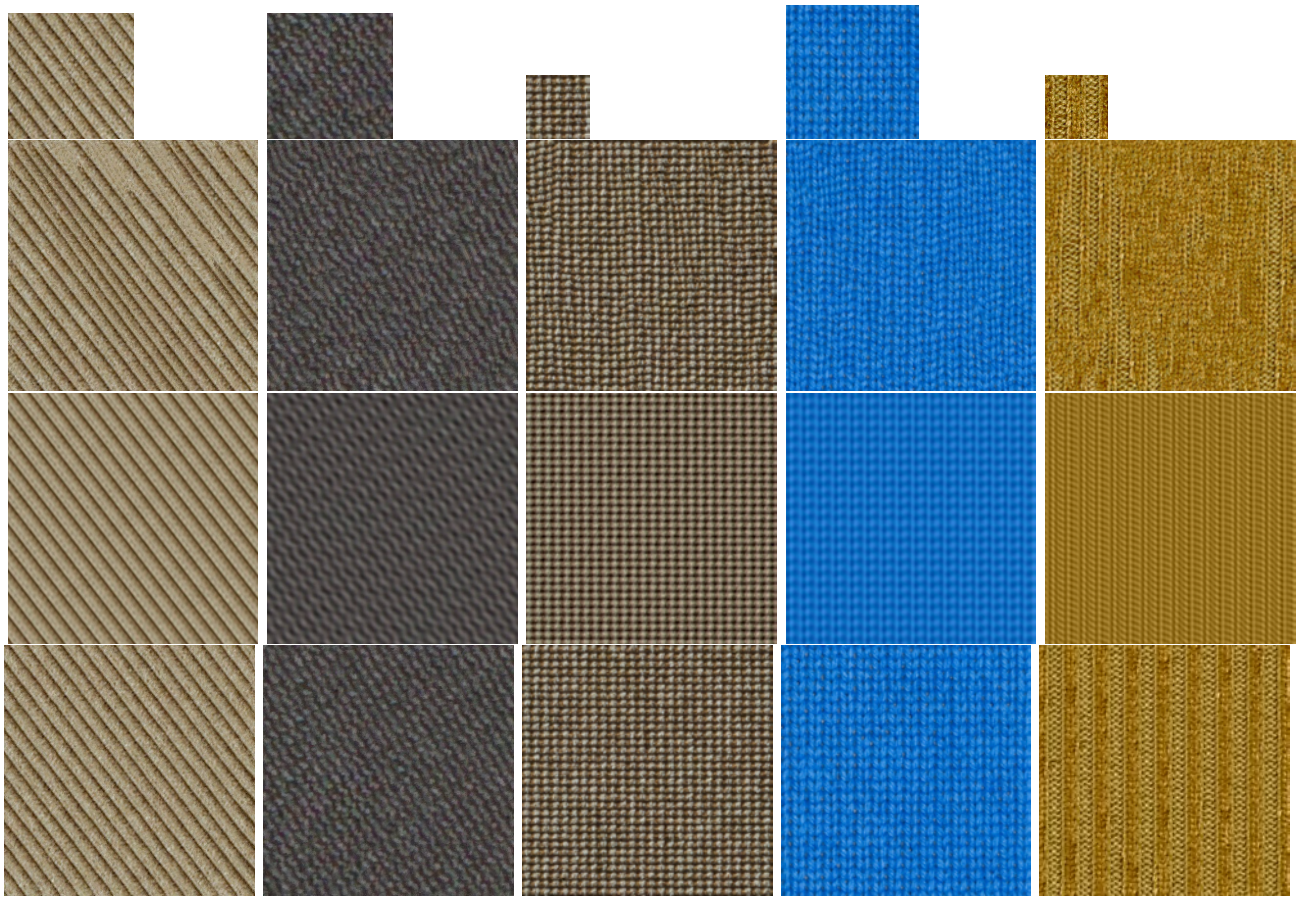


**Fig. 4.12: Examples for successful synthesis. Each column shows from top to bottom the input sample, standard ANN synthesis without a mask, the mask created with the FrDFT synthesis, and the ANN synthesis result with mask. In many cases the FFTM provides a very accurate approximation of the synthesized texture already.**

As the results in Figure 4.12 show, FFTMs clearly help to preserve regular structures. While standard ANN-accelerated per-pixel synthesis of the textile textures fails to retain the appearance of the textures – especially reproduction of the rightmost texture fails completely – simply employing the FFTMs as textures performs much better already, since they preserve the textures' most significant property: their regularity. Addition of irregular detail further improves the quality of the texture whereas the degree of improvement depends on the visual dominance of the regular structure. Especially textures with visually rich appearance like the knitted wool or the rightmost texture profit extremely. Further, very nice results not representing textiles are shown in Figure 4.13.

Synthesizing textures using the patch-based approach leads to very good results which are superior to the ones of the ANN-accelerated pixel-based method for near-regular textures. Figure 4.15 shows an example in which the pixel-based algorithm fails to add the irregular details but which is handled by the patch-based algorithm very well.

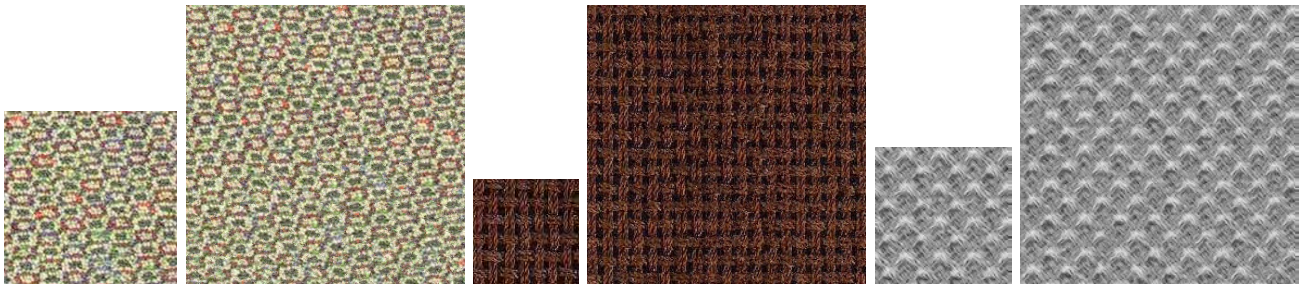


**Fig. 4.13: Examples for successful synthesis of non-textile samples with the pixel-based approach.**

Further results of this method are shown in Figure 4.14. Synthesis was done using the Graphcut algorithm and the sub-patch matching strategy [288], which was previously not known to give good results for near-regular texture, which confirms the usefulness of FFTMs. The used patch size can be determined automatically by computing parts of the texture that are tileable, which are usually much smaller than the full texture, leading to run-time advances compared to the entire patch matching strategy [288]. The determined patch size needs not be very accurate to achieve pleasing results (e.g., the bottom right texture in Figure 4.14 contains features of size $75 \times 60$ pixels and was synthesized using a patch size of $40 \times 40$). Please note that synthesis of the textures in Figure 4.15 and in the top left of Figure 4.14 was previously not possible using fully automatic algorithms [312]. Another advantage of using small patches is that textures with larger variance can be synthesized while still preserving the texture features representative for the input texture. This leads to visually more rich textures where repetition artifacts are less notable.

| | $128^2$ pixels | | $239^2$ pixels | |
|---|---|---|---|---|
| Neighborhood size | No PCA | PCA | No PCA | PCA |
| $2 \times 2$ | 0.16 | 0.07 | 0.42 | 0.10 |
| $3 \times 3$ | 0.42 | 0.12 | 1.44 | 0.14 |
| $4 \times 4$ | 0.90 | 0.21 | 3.22 | 0.21 |
| $5 \times 5$ | 1.65 | 0.37 | — | 0.30 |

**Tab. 4.4: Runtime per output pixel for the ANN synthesis in milliseconds for the first ($128^2$ pixels) and fourth ($239^2$ pixels) sample in Figure 4.12. PCA was applied to the neighborhood vectors from $T_{in}$ and $M_{in}$ to reduce their dimension to 16.**

Table 4.4 shows some runtime examples for the ANN synthesis, measured on a 2.4 GHz Pentium 4. The runtime increases with the neighborhood size, but can be reduced significantly using PCA. Other measurements show that the ANN search time depends on the sample appearance because the time varies for samples of the same size. In general, an approximate runtime of $O(\frac{|N|}{\epsilon} \log |T_{in}|)$ per output pixel was observed.

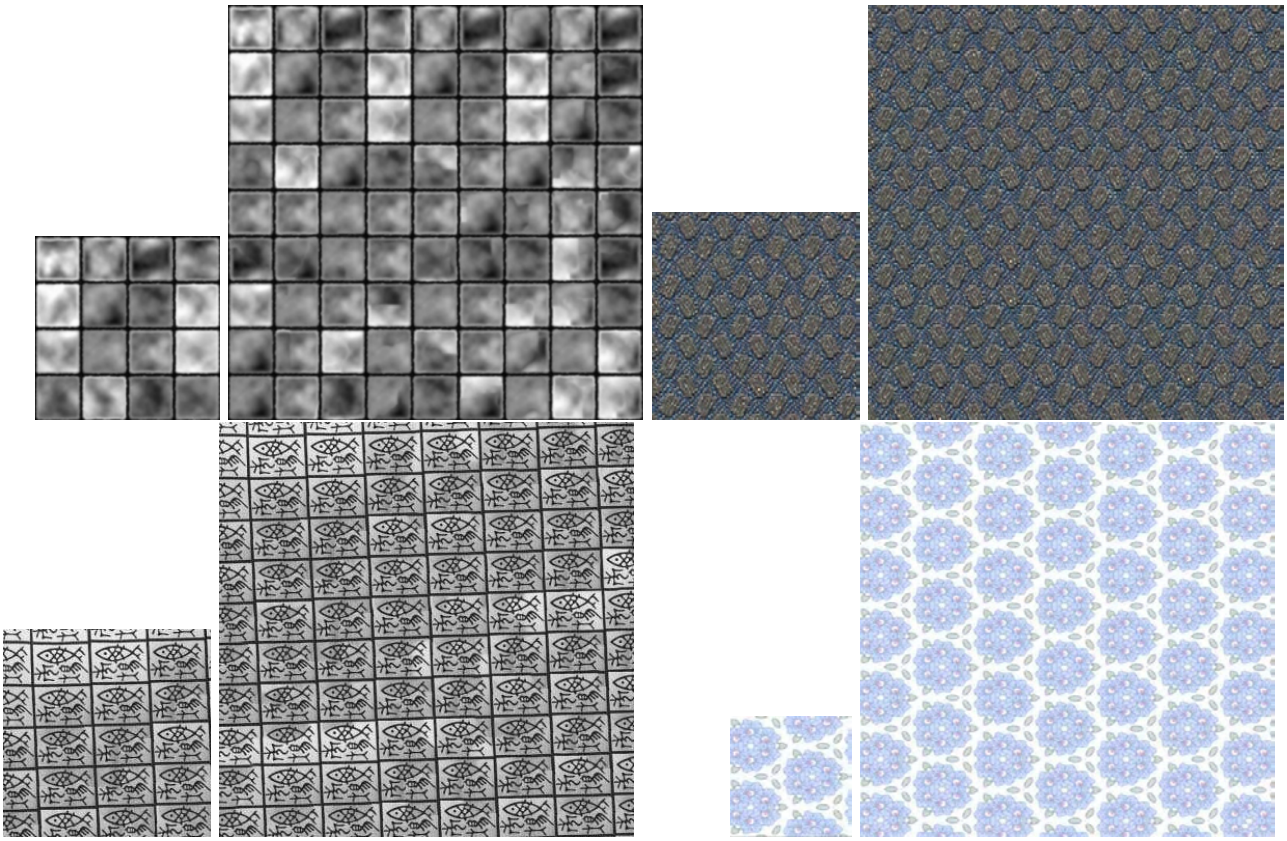


**Fig. 4.14: Examples for successful synthesis with the patch-based approach.**

Synthesis times for the patch-based method are longer, typically some minutes for the examples shown. The exact time naturally depends on the size of the synthesized texture, the number of improvement steps necessary or allowed, and the size of copied patches. Nevertheless, synthesis times are not noticeably longer than without incorporation of FFTMs.

Comparing the pixel- and patch-based synthesis approaches is difficult. For most textures, ANN-accelerated synthesis already yields very good results at a fast speed. Nevertheless, the patch-based algorithm turns out to be more robust: It provides good results even in cases where the pixel-based approach fails.

The quality of the FFTM has great influence on the result. If the mask contains the regular structures without errors, the synthesized textures are of high quality. Otherwise, the synthesis algorithms produce errors as shown in Figure 4.16.

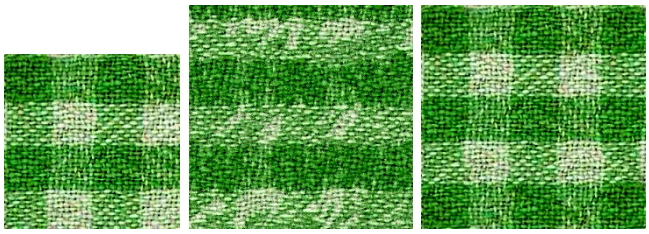


**Fig. 4.15: Example for improved quality of patch-based synthesis. Although the mask has good quality, the ANN synthesis (middle) fails to reproduce the irregular structures. Applying extended patch-based synthesis incorporating FFTMs (right) the results become very pleasing.**

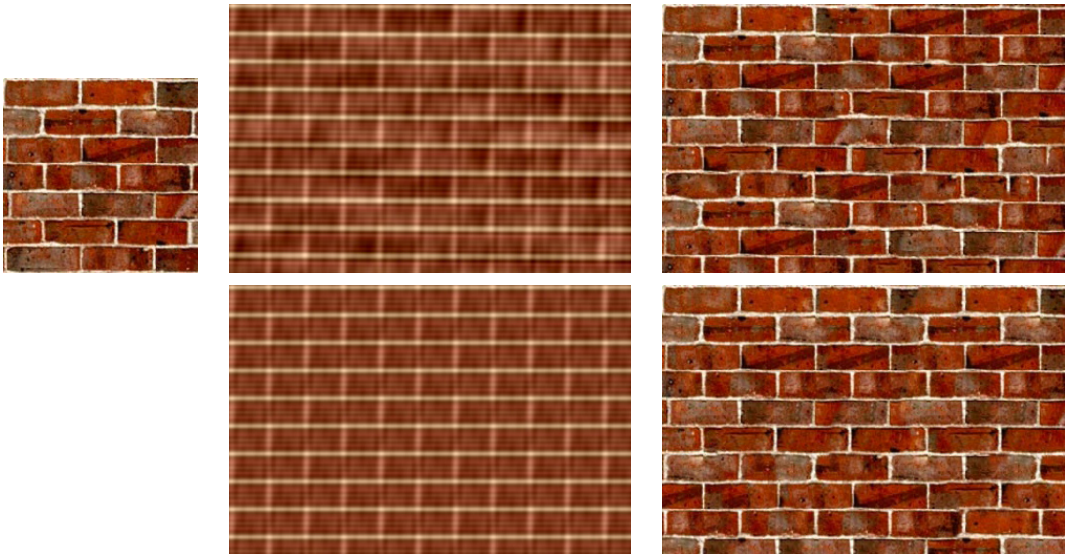


**Fig. 4.16: Problematic synthesis (top). The FFTM (middle) contains errors since the base frequency pair was not enforced. In such a case, texture synthesis (right) fails to reproduce the original appearance (left). The problem vanishes if enforcing the base frequency pair (bottom).**

### 4.1.6 Conclusions

In this section a new method for synthesis of near-regular textures was described. The key observation behind this technique is that an DFT intensity filter can be applied to separate dominant regular structures from irregular texture detail. Since the DFT intensity filter works for already tileable textures only, FrDFT was applied instead of DFT. An algorithm for automatic extraction, optimization and synthesis of the regular patterns was proposed, which allows to generate fractional Fourier texture masks. These masks are used in combination with existing sample-based texture synthesis algorithms to add the missing irregular texture details. The success of this approach was documented by showing high quality synthesis results for a number of textures from different material categories.

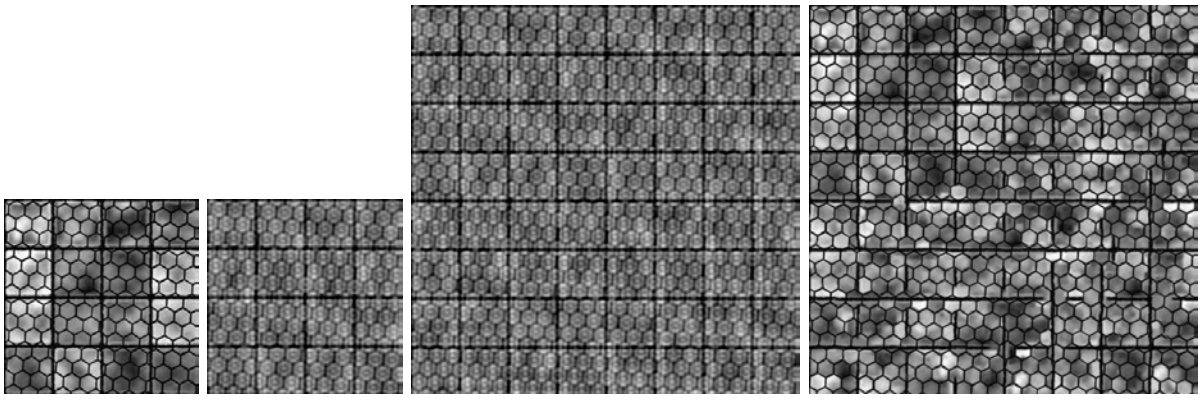


**Fig. 4.17: Problematic synthesis of textures with independent structures. The input texture (left) contains far less than two periods of texture. Although the extracted FFTMs (middle) capture the regularity, sample-based algorithms fail to preserve texture features (right).**

Application of the FrDFT intensity filter should prove beneficial in combination with other methods as well. Liu et al.'s [324] approach for computation of shape and size of tileable texture elements, which is based on autocorrelations, should be improved by removing irregular texture parts. Additionally, the extracted frequencies represent an explicit description of unique texture properties and might thus contribute to accurate texture recognition.

Especially interesting results could be achieved by combining FTTMs and parametric texture synthesis techniques since the combination would result in a very compact, parametric texture model. Thus, besides efficient storage, efficient on-the-fly texture synthesis would become available. Additionally, parametric texture models are especially interesting since they *learn* an explicit texture model and can thus synthesize unseen texture data. The strength of such an approach is shown in Figure 4.17. The input texture contains two structures: the black, hexagonal grating and the square tiles colored with different shades of gray. Fractional Fourier analysis succeeds in extracting these two structures and generated FFTMs show accurate replications of the regular parts. Unfortunately, intelligent sampling fails to reproduce texture details. Due to the slightly different orientation of the two structures, the input sample does not contain a full period of texture, prohibiting accurate texture synthesis due to missing samples. In contrast to that, parametric models should succeed in reproducing texture detail since they are capable of inferring unseen data from the given observations.

Another interesting application area for the FrDFT intensity filter might be specialized compression techniques. Previously, a technique separating textures into deterministic (i.e., regular) and non-deterministic (i.e., irregular) parts was proposed [495] but it handles deterministic features based on DFT analysis. As shown above, better results can be achieved for near-regular textures using FrDFT analysis.

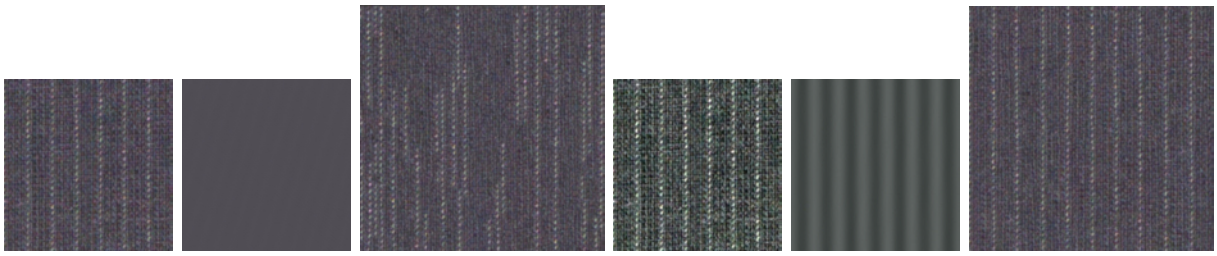


**Fig. 4.18: Shifting energy into regular parts leads to better FFTMs and thus synthesis results. From left to right: input texture, FFTM, synthesized result (pixel-based synthesis, $6\times6$ neighborhood), enhanced input (histogram stretch), enhanced FFTM, correctly synthesized result.**

Although the FrDFT intensity filter turns out to be very robust for textures with regular structures that carry a large amount of energy, handling of regular textures like pinstriped suits is difficult (cf. Figure 4.18). Fortunately, it is often easily possible to shift more energy into regular structures like the thin pinstripes by applying simple image processing operations (e.g., histogram stretch). Such an approach leads to similar texture masks than the ones created by the approach of Wu and Yu [592] which extracts sharp features using image filters.

Finally, in order to make the approach applicable to near-regular textures with substantial geometric variation or textures featuring significant perspective distortion, the method should be combined with regularization approaches like the one of Liu et al. [323].

## 4.2 BTF Synthesis

Synthesizing materials with texture synthesis algorithms as the one proposed in the previous section leads to textures that closely resemble their archetype. Despite the impressive results achieved by this approach, synthesized materials are insufficient for predictive rendering since they represent spatially varying diffuse BRDFs. Obviously, for predictive rendering more complex materials are required.

Chapter 2 already presented some methods extending texture synthesis methods to more general material representations like BTFs. Since synthesis techniques make no assumptions about the data they process, they can easily be applied to high-dimensional data sets like BTFs in principle. Unfortunately, the high dimensionality introduces practical problems by significantly increasing synthesis times. In special, computation of similarities of finely sampled ABRDFs requires much more effort than computing similarities of RGB color values. Therefore, the most crucial decision in extending synthesis algorithms to complex material representations is definition of an efficient method for computing similarities.

In the following, experiments with such methods are presented. The first experiment evaluates a pixel-based BTF method and provides interesting insights into problems of such an approach. The second experiment describes a novel combination of intelligent tiling and BTFs, leading to a method suitable for texturing very large surfaces without notable repetition.

### 4.2.1 Pixel-Based BTF Synthesis

The first experiment with BTF synthesis is based on the BTF synthesis on surfaces method of Tong et al. [520]. It consists of two steps. First, ABRDFs are clustered and similarities between cluster centers are precomputed. Then, BTFs are synthesized using a hierarchical pixel-based synthesis method, employing the precomputed data to achieve efficiency. Unlike in the original paper of Tong et al., the BTFs synthesized in this experiment are planar, thus complex resampling of BTFs on meshes is avoided.

#### Analysis Step

Synthesizing BTFs is problematic due to two main reasons: First, BTFs typically require several GBs of storage. Second, the high dimensionality of ABRDFs makes similarity computations very expensive. To reduce both problems, Tong et al. introduce a precomputation step that clusters the per-texel ABRDFs using the k-means clustering algorithm. In the test described here, 1000 to 2000 clusters were used, which yields a

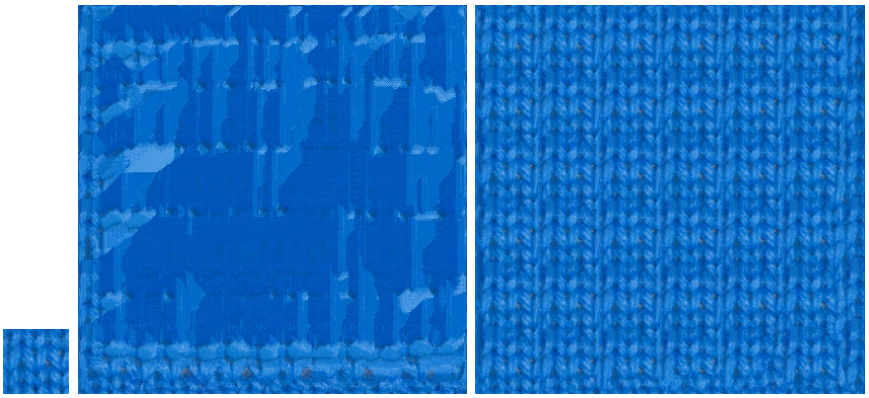


**Fig. 4.19: Pixel-based BTF synthesis results from knitted wool sample (left) using textons (center) and raw data (right).**

high-quality representation of the BTF. Since the number of clusters is relatively small, one can afford precomputing the similarity of clusters by calculating $L_2$ distances of cluster center ABRDFs. After these precomputations, BTFs can be represented by spatially varying cluster indices, leading to a compact representation, and efficient synthesis is possible by reusing precomputed similarities of cluster centers.

While the experiments confirmed that clustering leads to much more efficient processing while retaining the quality of synthesized results, a different preprocessing step suggested by Tong et al. turned out to reduce synthesis quality. The authors propose to handle vectors of 3D textons (i.e., responses to $n$ Gaussian filters) instead of vectors of color values (i.e., ABRDFs). Unfortunately, this approach seems to prohibit preservation of fine structures in the BTF (cf. Figure 4.19) and additionally increases the storage requirements by a factor of $n$.

To reduce necessary precomputation time, one can select a few hundred representative images and cluster them instead of all images representing the BTF (here: $81^2$). Experimental results show that this approach reduces synthesis quality just slightly while decreasing preprocessing time from about one hour for a full $64 \times 64$ pixels BTF to about ten minutes if 150 representative images are used (including selection time).

### Synthesis Step

The second step of Tong et al.'s method synthesizes a new BTF by a hierarchical, pixel-based scheme. Multi-resolution texture synthesis is made possible by considering BTFs at various resolutions (resolution in each spatial dimension is halved in each additional level) and performing preprocessing for each level. For each synthesized layer, standard pixel-based synthesis is performed, utilizing the precomputed similarities for efficiency. Additional speedup is gained by restricting candidate sets according to the $k$-coherence

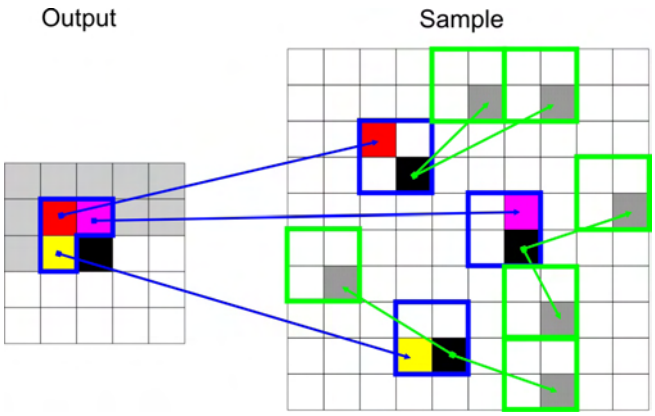



**Fig. 4.20: Candidate set for $k$-coherence synthesis when synthesizing the black pixel. The blue areas are the candidates for $k = 1$. For $k = 3$ for each of these candidates the two most similar neighborhoods are added as additional candidates (green).**

scheme. The idea of this methods is to find, for each pixel in the sample, a number of $k$ pixels with most similar neighborhoods (cf. Figure 4.20). The candidate sets can be precomputed before the actual synthesis step. As the approach of Ashikhmin [10], this technique favors copying connected regions, but it achieves greater randomness for $k > 1$. Other than traditional texture synthesis methods, the approach of Tong et al. does not copy color values into the output texture but texture coordinates referring to the input sample. This idea leads to a significant advantage: texture coordinates are resolution independent, thus allowing independent preprocessing of data per level.

When completing synthesis of one BTF level, the resolution of the synthesized texture is doubled in each spatial dimension. Information from the completed layer is reused in the new layer by copying texture coordinates as shown in Figure 4.21, initializing a quarter of all pixels. Since the lower resolution sample was created by bilinear interpolation, similar pixels at these coordinates will be found in the current layer. The initialized pixels are omitted in the following synthesis step. After synthesizing the missing pixels, synthesis is performed on the previously omitted pixels to reduce visible errors.

After synthesizing the highest resolution level of the BTF, the algorithm has generated an index texture where each pixel contains texture coordinates referring to the original BTF. Efficient rendering from this representation can be done in two ways. Either the BTF is compressed using some compression technique and the indices from the index structure are resolved during rendering, or BTF compression techniques using clustering like the one described in the previous chapter are employed, in which case the texture coordinates in the pixels can be replaced by cluster indices. Choosing either of the two techniques, very few additional memory is required for storing synthesized regions, making the approach very useful in combination with efficient, real-time rendering.

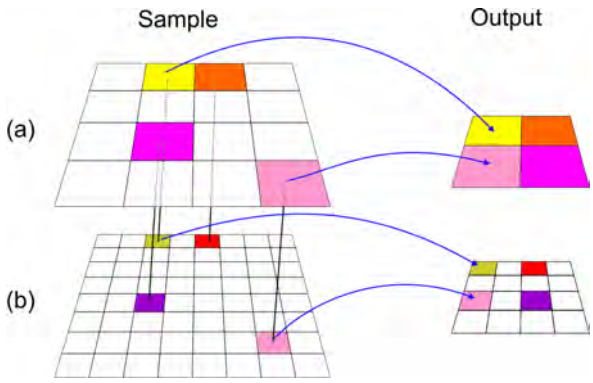



**Fig. 4.21: Hierarchical synthesis with two layers. (a) Lower resolution after synthesis. The pixels of the output store texture coordinates into the sample. (b) Higher resolution before synthesis. The texture coordinates from the lower layer are copied to the higher resolution and used as already synthesized pixels.**

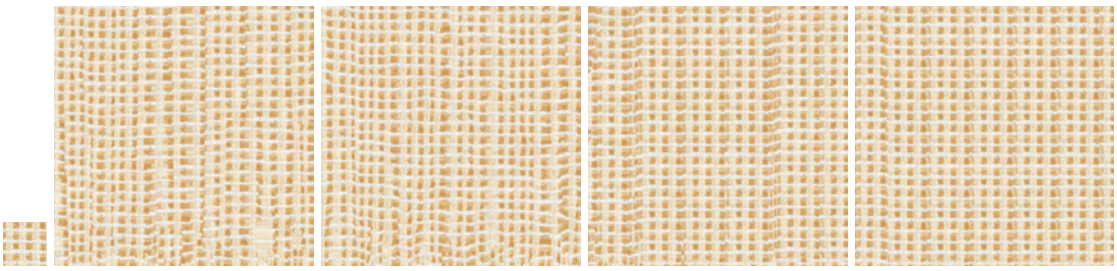


**Fig. 4.22: Synthesis results for a cushion fabric BTF with different parameters. From left to right: original, synthesized with neighborhood size $2 \times 2$, one layer, $k = 1$ and $k = 5$, and with three layers, $k = 1$ and $k = 5$.**

## Results

The quality of synthesis results depends on the structure of the input, the values for $k$, and the neighborhood size. In general, larger neighborhoods and hierarchical synthesis improve the results. However, for materials with irregular structures, e.g., many natural materials as in [10], smaller values of $k$ yield better results due to the larger spatial coherence of copied pixels. Figure 4.22 shows results synthesized with various parameters. As shown in the previous section, very good results can be achieved for most textures when choosing appropriate parameters.

Interestingly, synthesis results are not equally accurate for all combinations of light and view directions. While relatively simple samples like the cushion fabric show only hardly visible errors, problems become quite significant for materials with complex structures like knitted wool (cf. Figure 4.23). One reason for this problem is the following: BTF images captured at grazing view angles effectively feature a lower resolution due to area-foreshortening. Pixel colors are therefore assigned by linearly interpolating

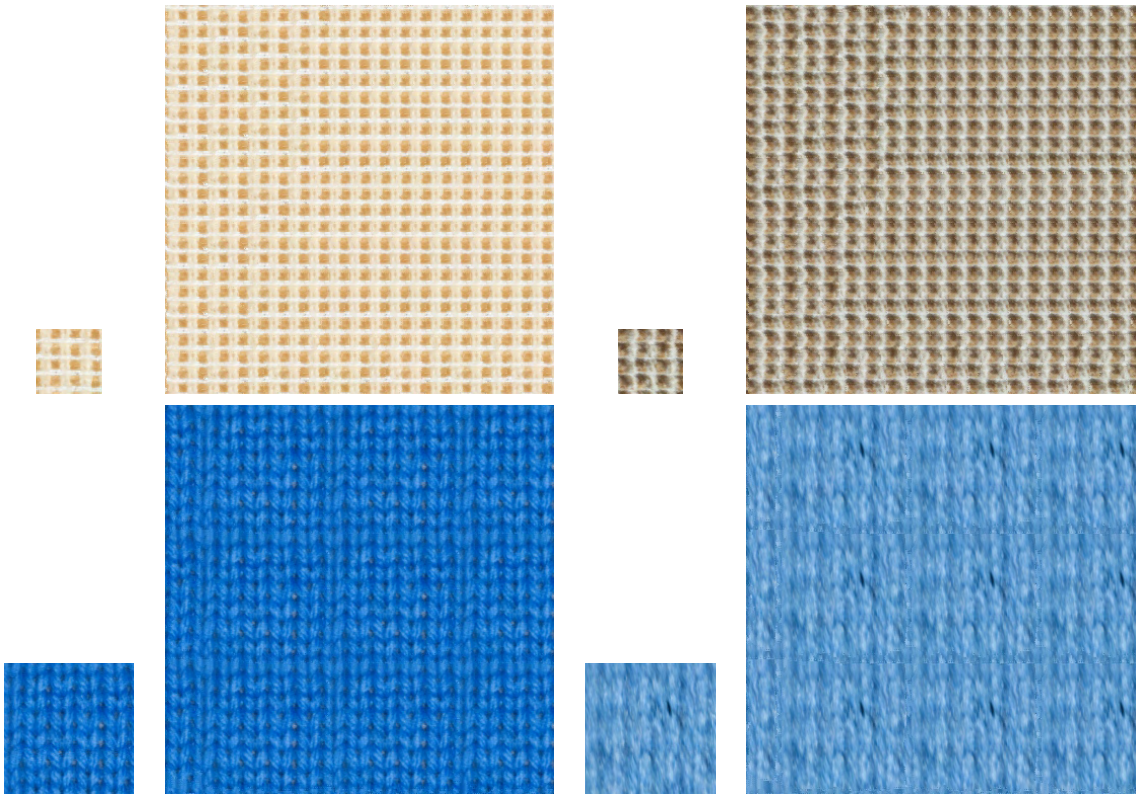


**Fig. 4.23: Synthesis results for different materials. Left: view and light direction orthogonal to the plane. Right: grazing view and light angles.**

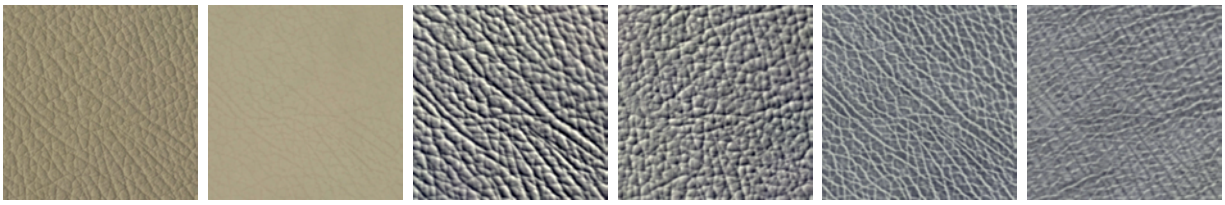
neighboring values. Obviously, this blurs out structures, reducing the dissimilarity of pixels compared to images from other view directions. Thus, these images are underrepresented in similarity computations which include contributions from all combinations of view and light directions with equal weights. Possibly, view-dependent weighting might improve this problem but unfortunately determination of such weights is very difficult. Therefore, the problem persists for the moment.

Compared to results from patch-based texture synthesis algorithms, the generated results show some noisy regions where color values do not seamlessly integrate into the surrounding structure (especially visible in Figure 4.22). These problems are mainly due to the fact that storing texture coordinates per pixel instead of color values of ABRDFs prohibits feathering of copied regions, which is commonly employed in patch-based synthesis algorithms.

Table 4.5 shows the required runtime for BTF synthesis with three layers for different output sizes and values for $k$, excluding the time for precomputing cluster similarities. Due to precalculation, the sample resolution and number of selected BTF images have no effect on synthesis time. The timings where obtained on a 2.4 GHz Pentium 4.

| Size | $256^2$ | $256^2$ | $256^2$ | $256^2$ | $512^2$ |
|---|---|---|---|---|---|
| $k$ | 1 | 3 | 6 | 6 | 6 |
| Neighborhood size | 2 | 2 | 2 | 3 | 2 |
| Time | 1.9 s | 2.7 s | 3.7 s | 13.6 s | 13.4 s |

**Tab. 4.5: Synthesis times with 3 layers.**



original mean 1st comp. 2nd comp. 3rd comp. 4th comp.

**Fig. 4.24: Full matrix factorization of leather BTF. Color values from component textures were scaled and centered at luminance value 128 to make them better visible.**

### 4.2.2 Tile-Based BTF Synthesis

Due to the superior synthesis quality of patch-based algorithms when handling color textures, another experiment concerning BTF synthesis was conducted. This time, a patch-based method was used to generate a set of texture tiles which fit each other seamlessly [492]. Using intelligent tiling approaches allows seamless texturing of large objects without introducing obvious repetitions of textual structures.

As in the approaches of Koudelka et al. [278] and Liu et al. [319], BTFs are factorized into Eigen-Textures (cf. Figure 4.24), and view- and light-dependent reconstruction weights. To reduce memory requirements, only the $k$ most significant Eigen-Textures are retained ($k = 100$ in the conducted experiment). Thus, the per-pixel dimensionality is reduced from $3 \cdot |\mathcal{M}_i| \cdot |\mathcal{M}_r|$ to $3k$ ($\mathcal{M}_i$ and $\mathcal{M}_r$ denote the sets of measured light and view directions). As the eigen-value plot in Figure 4.25 shows, this suffices to capture the most significant elements of the BTF.

Reducing the per-pixel dimensionality enables much more efficient similarity computations. In the given case, similarity is computed as the $L_2$ norm of the concatenated $3k$ per-pixel values. For small $k$ this can be computed very quickly.

Unfortunately, storage requirements for suitable sample BTFs are quite high: about 3.5 GB for high-dynamic range BTFs consisting of $512^2$ pixels (note: smaller BTFs turned out insufficient for generating tile sets due to limited variance of textual elements). Such memory amounts cannot be handled by in-core factorization algorithms on existing 32 bit operating systems. Therefore, implementation of an out-of-core data handling scheme became necessary. Due to the large matrix to be factorized and due to out-of-core data processing, factorization requires several hours on a single PC.

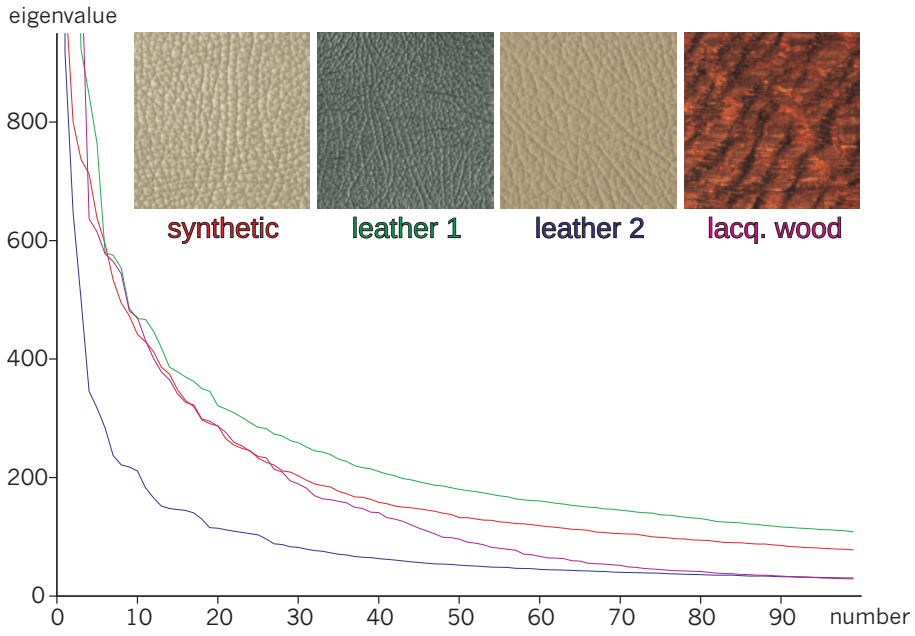



**Fig. 4.25: Eigenvalues for four BTFs.**

Synthesizing BTF patches follows the approach proposed by Somol and Haindl [492]. Based on a simplified, faster version of Image Quilting [122], texture tiles are generated by overlapping patches, determining minimum error boundary cuts, copying interior pixels, and blending pixels along the cut boundary. The size of generated tiles has to be chosen manually but can also be computed automatically for near-regular BTFs applying the technique described in the previous section. To enhance synthesis speed, only the six most significant Eigen-Textures are used during synthesis. Thus, similarity computations reduce to dot products of vectors with 18 components. Yet, synthesis times for a single material are still about eight hours.

Since high-quality BTF reconstruction from the six most significant Eigen-Textures is not possible, at each synthesized pixel the location of contributing pixels from the input texture together with respective blend weights are stored. Reapplying blending to the corresponding pixels from the high-quality BTF representation with 100 Eigen-Textures leads to high-quality BTFs without affording high-dimensional similarity computations during the synthesis process.

Figure 4.26 shows a tile set for a leather BTF generated with this approach. Obviously, the textural elements are preserved very well. Similar results were achieved for the other tested BTFs. As in the case of per-pixel BTF synthesis, the generated texture tiles feature problems for grazing view angles. Figure 4.27 shows two cases with visible cuts between overlapping patches. Compared to the pixel-based approach, the problems are less visible. This can be explained by the larger coherent regions of texture that are

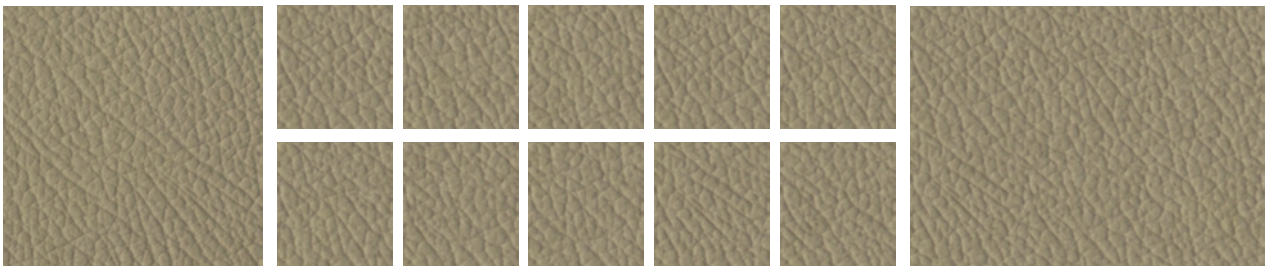
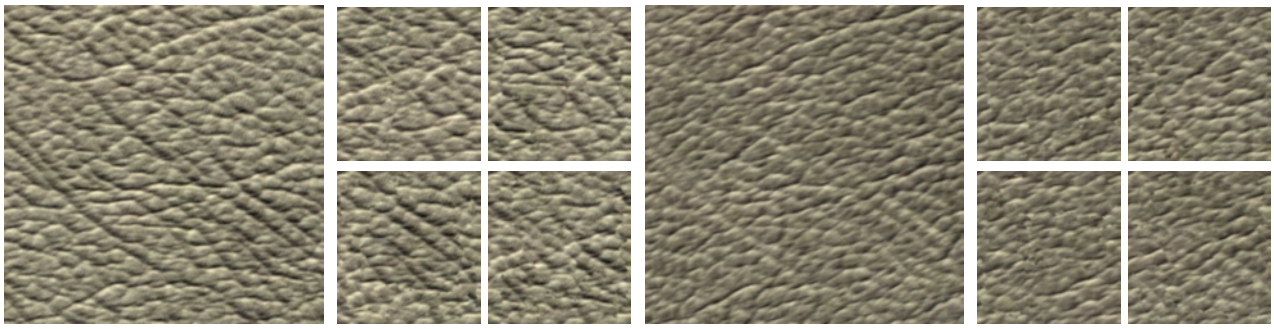


**Fig. 4.26: BTF tile set generated from leather BTF. Right: tiled plane.**



original tiles original tiles

**Fig. 4.27: For grazing view and light angles, synthesis problems become visible.**

copied to the output texture. Additionally, blending along boundaries reduces visible problems. Fortunately, the remaining errors are hardly noticeable during rendering (cf. Chapter 9).

The data used during the synthesis process is not suitable for efficient rendering, since too many Eigen-Textures need to be used to obtain high-quality results. Therefore, synthesized tiles need to be compressed with a method suitable for real-time decompression in a following step (e.g., compression based on Reflectance Fields). Especially high compression rates can be achieved in combination with clustering compression techniques if the same set of clusters is used for all tiles. In such cases, additional storage is only required due to the larger spatial extent of the tile set compared to the input BTF.

## Conclusions

As the results in this section show, the patch-based approach achieves very good quality even for complex input data with significant height variation or highly non-uniform spatial reflectance behavior. Compared to the pixel-based synthesis techniques, much better results can be achieved at the cost of longer processing times. Therefore, the patch-based approach is more suitable for predictive applications.

Despite the nice quality achieved, synthesized textures significantly deviate from input samples. Especially for grazing view angles, obvious cuts are visible. Fortunately,

these problems only occur for surfaces seen at grazing angles. Therefore, they are hardly visible when rendering scenes containing respective synthesized materials since texture filtering tends to smooth them out. Nevertheless, more accurate results should be achieved in the future, e.g., by introducing more suitable similarity measures for ABRDFs.

The methods for BTF synthesis tested in this section are currently state of the art. Problems reported for the tested methods are therefore common for most existing approaches. Techniques potentially avoiding these problems show significant other problems. Methods based on parametric texture models should be robust to grazing view problems but currently lead to insufficient synthesis quality due to inadequate reproduction of texture features for all combinations of view and light directions. A more promising direction is taken by the patch-based approach of Liu et al. [321] which handles mesoscopic BTF geometry independently. In a separate synthesis step, mesoscopic height maps are synthesized. Assuming an average BRDF, the authors derive shading values for each view- and light direction and use this information to guide faithful BTF synthesis. Unfortunately, they handle each combination of view- and light direction separately, leading to inconsistent BTF views due to randomized BTF synthesis. Additionally, using average BRDFs fails to model several spatially dependent material properties. Nevertheless, the underlying idea might improve the quality of synthesized BTFs if all shading values are considered concurrently. An alternative technique might directly employ absolute or relative height information for guiding texture placement without deriving shading values. Obviously, derivation of highly accurate height information would be required, either using photometric stereo approaches or by explicitly measuring height values [554].

## 4.3 Summary

Summarizing the results from this chapter and other publications, most kinds of textures can be handled by texture synthesis methods in a satisfying manner today. While some techniques focus on special types of textures, others handle a broad range of inputs. Yet, none of them handles all kinds of textures in an optimal way.

To make texture synthesis more easily usable in industrial applications, commercial texture synthesis tools should be developed. They should feature the possibility to transform acquired photographs into images which can serve as input into texture synthesis algorithms. Necessary steps could be removal of perspective distortion and light variation, and possibly some way for enforcing planarity of the acquired sample. Such tools should additionally guide the user in selecting the most suitable synthesis technique for a specific input texture. Optimally an automatic method would select the best available technique by analyzing the input (e.g., using measures computing properties like regularity [323]).

Surface materials are obviously not limited to color textures. The results shown in this chapter demonstrate that synthesis of more complex materials is also possible. Nevertheless, they also reveal that several problems remain when handling BTFs, which will likely occur for other complex material representations as well. Most of these problems are due to the much higher dimensionality of representations. Future methods should devise new concepts for accurately handling these dimensions, e.g., by identifying most relevant parts. While existing methods employ similar concepts already, data reduction methods like PCA do not guarantee to capture parts that are perceptually relevant.

The previous judgments of texture synthesis quality are solely based on subjective assessments. The same holds for published studies comparing or evaluating the synthesis quality of several algorithms. Lin et al. [312] compared the results of several techniques when applied to near-regular textures. Dong and Chantler [112] assessed the suitability of five synthesis methods for rough textures (i.e., combinations of bump and color texture maps, or BTFs) using psychophysical studies. Obviously both papers focus on very specific material types and are based on subjective preferences. To generate objective results, comparison metrics would be required. Yet, automatic evaluation of synthesized materials is extremely difficult since one-to-one comparison against ground truth data is not possible. Such metrics therefore would be required to know the stochastic process generating the materials and would thus represent the basis for perfect material synthesis algorithms. In summary, it seems that numerous research opportunities remain in the area of validation of material synthesis algorithms.

# Part III

# Level of Detail Representations for Geometry

# Chapter 5

# State of the Art

In the previous parts of this thesis, it was pointed out that accurate scene modeling plays a crucial role for achieving predictive rendering: If the objects in rendered scenes lack the required accuracy, rendering techniques obviously cannot compensate this. The previous chapters presented or referenced methods for acquiring or modeling, and accurately and efficiently representing surface materials and – very briefly – light sources. The remaining ingredient for accurate scene modeling, the geometry of objects in the scenes, will be the topic of this third part.

As mentioned previously already, geometry is handled in a different way than material and light: It is not measured but modeled since this reflects the standard way in Virtual Prototyping. Geometric modeling is not limited to designing the shapes of the scene objects but additionally includes various steps that map CAD data to representations useful for efficient rendering. These steps include repair of inconsistent triangulations and inconsistently oriented patches, fixing problems with invalid normals, application of materials to geometric surface, and many others.

Due to the required accuracy of modeled geometry, the results of these steps are very complex: either models consisting of tens or hundreds of thousands of trimmed Non-Uniform Rational B-Spline Surfaces (NURBS) (which are the de-facto standard for surface modeling in industrial applications) or triangulated versions comprising several millions of triangles [106]. Despite the tremendous capabilities of existing graphics boards, GPUs are currently not capable of handling such models in real-time[1]. Even worse, some of these models do not even fit into main memory [106], requiring out-of-core strategies. Therefore, to enable (efficient) rendering of such models, it is essential to reduce the number of rendered surfaces and the number of surfaces kept in main memory.

[1] Alternative rendering methods based on ray-tracing can handle such models [538] but are currently not able to achieve real-time performance for industrially relevant display resolutions on standard hardware configurations.

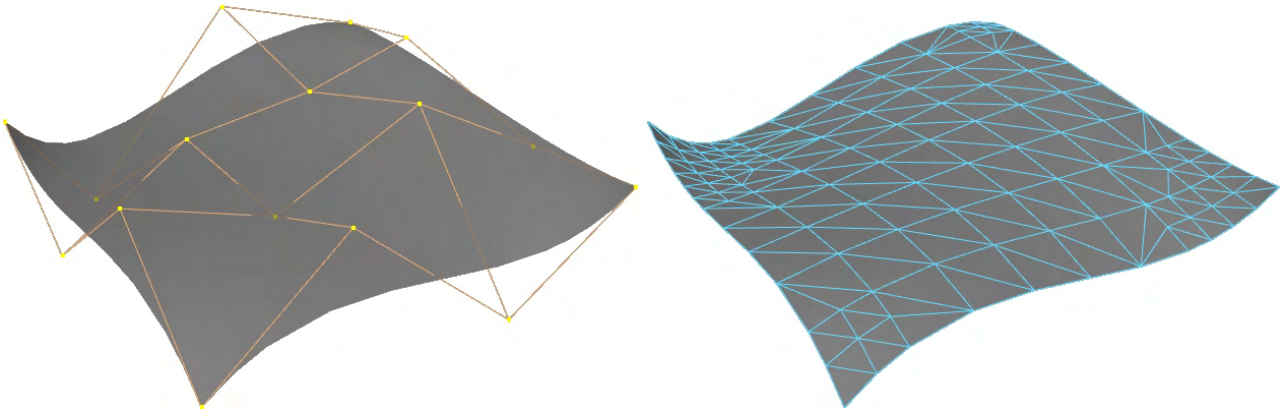


**Fig. 5.1: A simple NURBS surface with the net of control points (left) and its tessellation (right).**

To achieve this goal, two main approaches exist.

- *Visibility culling* techniques try to determine the set of visible surfaces at a very early stage of or even before rendering, enabling the rendering technique to focus its efforts on the surfaces contributing to the final image. Since exact solutions to this problem require significant computation time, practical approaches (cf. [33, 32]) restrict themselves to determining approximations of the exact set of visible surfaces. To achieve predictive rendering quality, especially *conservative* visibility culling techniques are of importance, since they never classify a visible object as invisible. Thus, they avoid image errors due to incorrect culling.

- Approaches from the second category, the *Level of Detail* (LOD) techniques, aim at adjusting the complexity of displayed models to the display size of the objects and the resolution of the imaging device. The benefit of these methods is twofold: First, they reduce the complexity of distant objects, leading to greatly increased rendering speed. Second, they eliminate aliasing artifacts since simplified objects (including all appearance attributes) are adjusted to the display resolution.

In the following, this thesis will restrict itself to description of LOD techniques. Interested readers find an overview of visibility culling techniques in the recent survey of Bittner and Wonka [33] and in recent publications focusing on occlusion culling [32, 279].

Due to the focus of this thesis on predictive rendering of digital prototyping data, covered LOD approaches are limited to those handling geometric surface types relevant for industrial applications. Separate sections will be devoted to LOD representations of models consisting of trimmed NURBS surfaces and to triangle meshes.

# 5.1 LOD Representations for NURBS Models

Trimmed NURBS surfaces are the most commonly used surface type for modeling in industry. Their large importance is due to the fact that almost arbitrary surfaces can efficiently be modeled by trimmed NURBS surfaces. Modeling NURBS surfaces

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{n} R_{i,j,p,q}(u,v) P_{i,j} \quad 0 \leq u,v \leq 1$$

is relatively easy and intuitive since their shape is mainly controlled by a net of control points $P_{i,j}$ (cf. Figure 5.1), where typically $P_{i,j} \in \mathbb{R}^3$, and the degrees $p$ and $q$ of the surface in directions $u$ and $v$, which influence the piecewise rational basis functions $R$ (for a definition see [414], page 128). Often, only specific regions of NURBS surfaces are active. Typically, these regions are defined by trimming curves (B-Spline curves [414] or piece-wise linear line strips) which restrict the range of valid parameter pairs $(u,v)$.

In contrast to their excellent suitability for modeling, trimmed NURBS are ill suited for rendering on existing graphics hardware, which handles point, line or triangle primitives only. Therefore, NURBS have to be converted into sets of these primitives at some point before rendering. Although approaches exist that convert NURBS into sets of points [65], most approaches rely on triangles. The conversion process triangulating the NURBS surfaces is referred to as *tessellation* and plays a key role for visualization of CAD models.

## 5.1.1 Trimmed NURBS Tessellation

Tessellation algorithms for trimmed NURBS surfaces have to fulfill several constraints. Obviously, they need to produce valid triangulations, i.e., triangulations that approximate the parametric surfaces with a maximum prescribed error according to some suitable metric. Additionally, they need to be efficient in terms of memory requirements and processing time. Finally, they should produce triangulations optimized for rendering, i.e., a small number of well-shaped triangles[2].

Due to the conflicting requirements, a large number of tessellation algorithms have been proposed. All of them first convert the trimmed NURBS surfaces into Bézier patches [138] and corresponding trim curve segments. Then the trimmed Bézier patches are evaluated at a number of sample points and the resulting 3D vertices are triangulated[3]. Existing algorithms especially vary in the implementation of the sampling

[2] Long, thin triangles may cause shading artifacts due to interpolation of normals.

[3] Typically triangulation is performed in the 2D parameter domain.

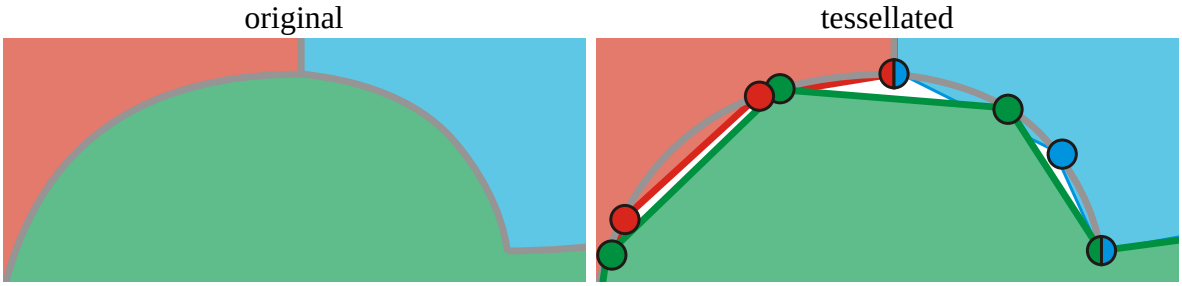



**Fig. 5.2: Cracks are introduced when tessellating adjacent patches (red, green, blue) individually without considering the placement of vertices of adjacent patches on common trimming curves (gray).**

and triangulation steps. Roughly, they can be divided into techniques placing samples at uniform locations [439, 1, 285], which leads to suboptimal triangulations but fast processing times, and others using adaptively, optimally chosen sample locations [151, 273, 17], which produce better triangulations at slower rates. Adaptive sampling techniques usually choose sample locations that minimize the geometric approximation error induced by replacing the trimmed NURBS surface with a triangulated version. Interestingly, most of them approximate geometric error from parametric deviation, and only few of them actually measure Euclidean error in 3D space. A notable exception is the approach of Guthe et al. [183] which additionally takes normals into account. This metric is especially interesting for rendering applications since it minimizes the shading differences between the original NURBS surface and the triangulated approximation.

Approaches suitable for real-time rendering should further be fast in order to enable on-line triangulation. Numerous such approaches were proposed [439, 285, 273, 64, 189, 18, 183], distinguishing themselves by processing speed, and the quality and complexity of generated triangulations.

Another criterion for tessellation methods suitable for rendering applications is their ability to produce crack free surfaces. Cracks are introduced in triangulations if trimmed NURBS surfaces are processed independently of each other (cf. Figure 5.2). During rendering, these cracks may lead to disturbing shine-through artifacts and should therefore be avoided at all costs. Existing publications try to solve this problem by choosing identical sample points on the trimming curves of neighboring patches [439], by sewing tessellated meshes [286, 21] or high-resolution line-strip approximations of trimming curves [189], or hiding these gaps in rendered images [18].

### 5.1.2 LOD for Trimmed NURBS Models

High-quality tessellations of models consisting of trimmed NURBS surfaces commonly contain millions of triangles. Unfortunately, this leads to two problems: First, rendering such large numbers of triangles hampers real-time rendering. This problem can be reduced using LODs of the model, derived from tessellating the models with varying quality criteria, at the cost of additional storage requirements. Second, this approach prohibits rendering the surface at arbitrarily high resolutions since the original surface representation is discarded.

To solve both the storage requirements and the resolution problems, researchers proposed to generate tessellations on-the-fly [439], with view-dependent tessellation parameters. Following this approach, only the triangles currently rendered need to be stored and the display resolution can be adjusted arbitrarily. Unfortunately, this leads to a different problem: CAD models commonly consist of several thousand (trimmed) NURBS surfaces. Converting these into Bézier patches and triangulating them into at least two triangles each leads to triangle meshes with several hundred thousand triangles. While these numbers can be reduced significantly using mesh simplification algorithms, such approaches require extensive processing time, prohibiting on-line tessellation. Therefore, the number of triangles needs to be reduced by other LOD schemes.

Kumar et al. [286] proposed to hierarchically merge NURBS surfaces into *super-patches* using region growing techniques. Super-patches are modeled as fixed degree NURBS surfaces. Thus, less Bézier patches are required to represent one super-face than two NURBS patches. In the proposed rendering method, the super-patches are pre-tessellated in advance and at rendering time, the suitable LODs are displayed. To prevent visible gaps between super-patches, an on-line stitching algorithm is used. Unfortunately, the stitching operations are very time consuming and error prone.

A different approach was presented by Gopi and Kumar [167]. They first convert the initial model into triangular Bézier patches using surface fitting techniques. Then, the new representation is simplified using vertex and face removal techniques traditionally applied to triangular meshes (see following section). The approach leads to nice LODs for models consisting of NURBS models but has problems to preserve the smoothness of input models. Additionally, it fails to handle trimmed surfaces. Surface fitting techniques were also used in the recent publication of Guthe et al. [184] for merging adjacent Bézier patches.

Sheffer [472] proposed direct simplification of CAD models consisting of parametric surfaces by selectively removing patches. The selection criterion considers changes to the region curvature, boundaries and surface areas resulting from removing a patch but includes no explicit measure of geometric error. Therefore, no error guarantees can be

given. In addition, it prohibits simplification of edges shared by more than two NURBS patches, which occur frequently in industrially relevant data sets.

In Chapter 6, a method similar in spirit to the one of Sheffer is presented, which selectively removes parametric patches. In contrast to Sheffer's approach, it handles arbitrary edges and guarantees accurate error bounds, making it suitable for application in predictive rendering applications.

### 5.1.3 Run-Time Management of Trimmed NURBS Models

Rendering trimmed NURBS models follows either of two paths. The standard approach simply renders pre-tessellated geometry. Rendered images typically look great but usually far more triangles than necessary are rendered, significantly impacting rendering speed. In addition, the approach prohibits arbitrarily close inspection of surfaces without visible errors.

Therefore, approaches from the second category generate rendered geometry on the fly, in a view-dependent manner (e.g., [439, 286, 64, 189]). The approaches rely on tessellation methods achieving excellent tradeoffs between tessellation speed and the complexity of generated triangulations [18, 183]. To minimize the tessellation overheads, only visible patches are processed. Visible patches are determined using view-frustum and backface culling algorithms operating on (groups of) NURBS patches. To avoid the substantial overheads due to trimming, Guthe et al. [184] proposed to generate triangulations of untrimmed patches and to perform the trimming using graphics hardware. The approach achieves excellent results for moderately complex models with negligible precomputation times, making it useful for inclusion in CAD modeling systems. Its success is tightly coupled with the gap filling algorithm of Balázs et al. [18]. Instead of closing gaps between individually tessellated patches using on-line stitching techniques, it hides them during rendering with negligible overhead. This enables much faster visualizations.

### 5.1.4 Discussion

Despite the availability of highly efficient rendering techniques for trimmed NURBS based on on-the-fly tessellation, existing systems typically rely on pre-tessellated data. Thus, they sacrifice the ability to display the models with arbitrary accuracy. The reason for this discrepancy between available and employed techniques are the significant overheads for on-line tessellation which often outweigh the increased rendering times due to over-tessellated models. Even the highly efficient technique of Guthe et al. [184]

only provides real-time framerates for models containing a modest number of trimmed NURBS patches.

To make on-line tessellation approaches useful for industry, it is therefore indispensable to reduce the number of patches that need to be tessellated in every frame. Chapter 6 presents a technique that achieves this goal by introducing the Seam Graph data structure, which enables selective removal of patches which are too small to be visible while maintaining connectivity information, which is essential for gap closing.

## 5.2 LOD Representations for Triangle Meshes

While NURBS surfaces are the de-facto standard for modeling geometric surfaces in industry, they feature two main disadvantages that hinder their use in rendering applications. First, as discussed above, rendering of trimmed NURBS surfaces is not directly supported by graphics hardware. Therefore, they are typically converted into a representation that can be rendered more efficiently. Second, fine-grained assignment of materials to NURBS surfaces is very complex. Typically, spatially invariant materials are assigned on a per-patch basis. Obviously, this is more restrictive than assigning materials on a per-triangle basis. Additionally, lack of spatial variation prohibits a wide range of real-world materials.

Both restrictions do not apply to triangle meshes, since triangles are the rendering primitive existing graphics hardware is most optimized for, and since fine-grained assignment of spatially varying materials is possible. Following the definition of Botsch [44], triangle meshes are composed of a set of vertices $\mathcal{V}$ and a set $\mathcal{T} \subset \mathcal{V} \times \mathcal{V} \times \mathcal{V}$ of triangles, where vertices are associated with points in space – for rendering applications Euclidean 3D space. Additional entities in meshes are the edges $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ which bound the triangles and connect pairs of vertices. An important property of triangle meshes influencing the efficiency of methods operating on them is *manifoldness*. A triangle mesh is two-manifold if it is locally homeomorphic to a disk, i.e., if no edge is adjacent to more than two triangles, no vertex is adjacent to more than one fan of triangles, and the triangles do not intersect each other [44].

In industrial applications, triangle meshes are typically derived by tessellating trimmed NURBS models. They therefore represent piecewise-linear approximations of higher-degree surfaces, with the geometric approximation error being inversely proportional to the number of triangles in the mesh [44]. The complexity of triangle meshes can be adjusted by inserting of removing faces. Usually, LOD representations are derived by simplifying an existing mesh.

In the following, the four components of such LOD schemes are presented. First,

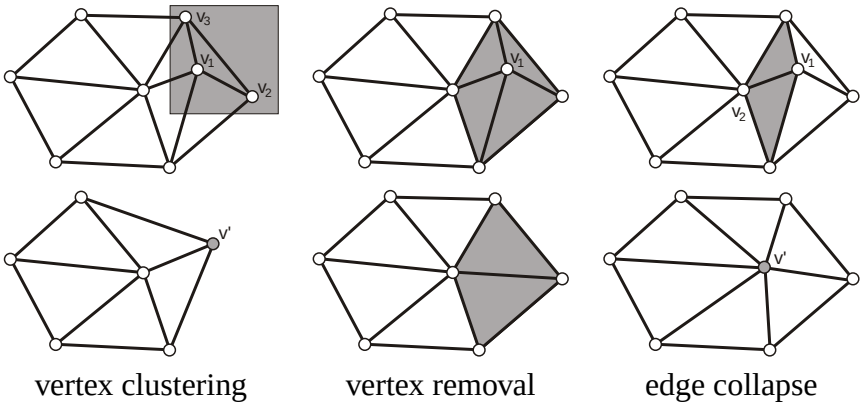



**Fig. 5.3: Local simplification operators. Top: before simplification, bottom: after.**

techniques for simplification of triangles meshes are overviewed. Then, metrics guiding these simplification operations are presented. Next, data structure allowing for efficient management of LOD representations are reviewed. Finally, techniques for real-time selection and rendering of suitable LODs are discussed.

### 5.2.1 LOD Operators for Triangle Meshes

LOD representations of triangle meshes are typically derived by simplifying an input mesh, which corresponds to reducing the number of triangles of the mesh. Over the years, several approaches towards this task were developed, which can roughly be categorized into two groups:

- Iterative approaches apply a series of local simplification operations to the mesh.
- Replacement methods replace the input mesh by another geometric representation.

The replacement methods can further be divided into methods yielding new triangle meshes (remeshing techniques) and those employing less complex geometry types augmented with image-based data (image-based techniques). In the following, existing research on these simplification operations is reviewed.

#### Iterative Mesh Simplification

Iterative mesh simplification methods apply series of local simplification operations to an input mesh that reduce its complexity. Typically, they decrease the number of triangles of a mesh, but they may also reduce its genus or the number of disjoint components. Over the years, a large number of such simplification operations were proposed, among

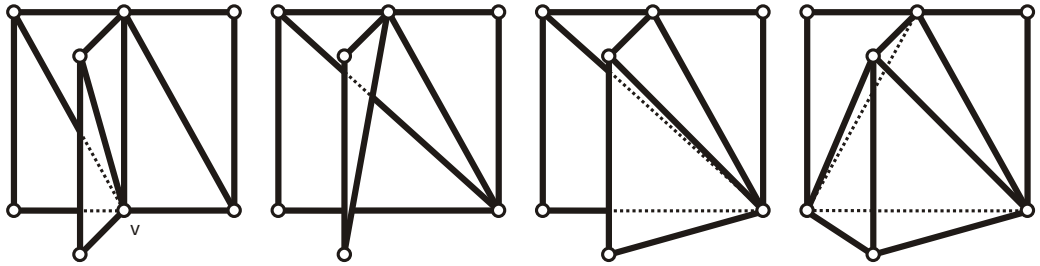



**Fig. 5.4: Ambiguities when refilling holes after removal of non-manifold vertex $v$.**

which the most prominent ones are vertex clustering, vertex removal, and edge collapse (cf. Figure 5.3).

Vertex clustering [441] merges all vertices in a specific region into a single vertex and removes degenerate triangles. The position of the new vertex is typically chosen such that an appropriate error metric is minimized. Application leads to arbitrary changes in the topology of the mesh, making the operation applicable to non-manifold and unconnected meshes. Vertex clustering removes an arbitrary number of triangles in a single step, making the operation very general but difficult to control.

Vertex removal [461] deletes a vertex and all adjacent faces from the mesh. Resulting holes in the mesh need to be filled by retriangulation of neighboring vertices. Due to ambiguities arising from application of vertex removal to non-manifold meshes (cf. Figure 5.4), this simplification operation is typically applied to manifold meshes only. A single operation removes either one or two triangles of a manifold mesh, depending on whether the removed vertex is located on the boundary. Therefore, the operation is very fine-grained, allowing for carefully controlled modification of meshes.

Probably the most widely used simplification operation is the edge collapse [225] which merges two vertices connected by an edge. The position of the new vertex can be chosen arbitrarily, or as one of the positions of the merged vertices. In the latter case the operation is called a halfedge collapse. Another edge collapse variant – the vertex pair contraction [156] – additionally allows to merge vertices that are not connected by an edge but are spatially close. This relaxation enables better handling of topologically complex surfaces since holes can be closed and disconnected parts can be joined.

Recent simplification systems are most often based on vertex clustering or variants of edge collapse due to the following reasons. Vertex clustering is very simple to implement, handles arbitrary triangle meshes, and naturally lends itself to applications where the mesh does not fit into main memory since only vertices belonging to a specific cluster need to be maintained in-core. In contrast, edge collapse operations are less simple to implement but are easily inverted, allowing for incremental refinement of simplified meshes [221]. Additionally, they typically lead to very high quality LODs.

Besides the three simplification operators described above, numerous others have been proposed [329]. In principle, all combinations of triangles, edges, and vertices can be collapsed [42], and respective removal operations are possible. A very interesting, alternative simplification operator was proposed by Popović and Hoppe [419]. Similar to the vertex pair contraction operation, it merges pairs of vertices. Yet, it allows introduction of degenerated triangles, i.e., triangles that degenerate to lines or points. Interpreting lines as cylinders and points as spheres, this introduces volumetric primitives that approximate the original geometry well and that can be rendered efficiently.

Hardly surprisingly, the order of simplification operations applied to the input mesh has a significant impact on the accuracy of simplified models. Since an optimal order cannot be determined efficiently, simplification algorithms typically follow a greedy approach: Operations introducing the least error according to some error metric (see below) are applied first. Typically, this is implemented using a global priority queue of simplification steps, which leads to very good results at the cost of managing such a queue. This overhead can be reduced using the technique of Wu and Kobbelt [589] which selects the best possible operation from a small set of potential candidates, where the suitability is evaluated on the fly. Interestingly, the quality of results achieved with this approach is just slightly worse, while the processing time and memory overheads can be reduced significantly.

Due to their greedy nature, iterative simplification schemes can typically be implemented relatively easily. In addition, they handle a wide variety of input models and produce good results in short times. Therefore, they have become the most widely used technique for large models. Unfortunately, the greedy nature of these algorithms cannot guarantee optimality of the simplified model, especially at drastic simplification rates (i.e., after applying a long series of simplification steps).

### Remeshing Techniques

In contrast to iterative simplification techniques, remeshing approaches handle the complete input model in a single step. They generate a new mesh that approximates the original according to some metric. Mostly, the vertices of this new mesh are derived by adequately sampling the mesh surface [525, 4], placing more points in regions with high curvature and less points in almost planar regions. Alternatively, the vertices can be derived from multi-resolution wavelet representations [120], which requires fitting a subdivision surface to the original geometry [39, 342], yielding an explicit LOD representation of the model. Closely related to this topic, researchers proposed to represent geometric objects as images, called *geometry images* [178, 446]. In these images, pixels correspond to vertices (and thus store 3D positions) and $2 \times 2$ pixel neighborhoods

define connectivity. LOD representations of geometry images can be derived using MIP-mapping techniques

Other remeshing approaches for meshes are based on converting the surface of an object into an implicit representation and tessellating it according to some criterion [210, 474]. Finally, some methods fit simple surfaces to the object (e.g., planes [250, 74], spheres, cones, and cylinders [591]) using region growing techniques or optimization approaches based on the Lloyd algorithm [325]. These simple objects can either be rendered directly, or they are joined and the union is triangulated.

The strength of remeshing techniques is their global handling of input models. They are capable of producing meshes that achieve an optimal balance of approximation error and mesh complexity. Unfortunately, they usually require much longer processing times than iterative simplification approaches. Additionally, global handling is restricted to meshes fitting into main memory. Therefore, out-of-core remeshing technique feature similar problems than iterative simplification approaches.

In the future, automatic derivation of subdivision surface representations from input meshes will be of special interest due to the inherent LOD nature of subdivision surfaces. Unfortunately, due to inherent geometric, topologic, and connectivity constraints of subdivision surfaces, it remains questionable whether such approaches are suitable for all types of meshes.

### Image-Based Object Representations

Image-based object representations [242] store some or even all properties of objects as images. A well-known and widely accepted image-based representation (IBR) are textured surfaces, where the texture encodes geometric properties of the surface that are too small to be modeled explicitly. Clearly, when viewing an object at various resolutions the amount of geometric detail efficiently encoded by textures will vary: At close inspection, almost all details need to be modeled explicitly. At far distances, most geometric details can safely be stored in a texture, reducing the explicitly modeled geometry to a minimum. Therefore, IBRs are especially suitable for distant objects. Another advantage of IBRs is that they allow for arbitrary topological simplification, since holes in objects can be represented as transparent pixels.

Accurate image-based object representations were first proposed by Maciel and Shirley [332]. They suggested to replace distant geometry by a textured quad which is oriented towards the viewer, the texture showing an image of the object as seen from the viewer's current position (cf. Figure 5.5). Such a representation is highly efficient for rendering large numbers of similar, distant objects (e.g., trees in terrain visualization). Unfortunately, the view-dependent nature of the texture leads to well-visible artifacts if the

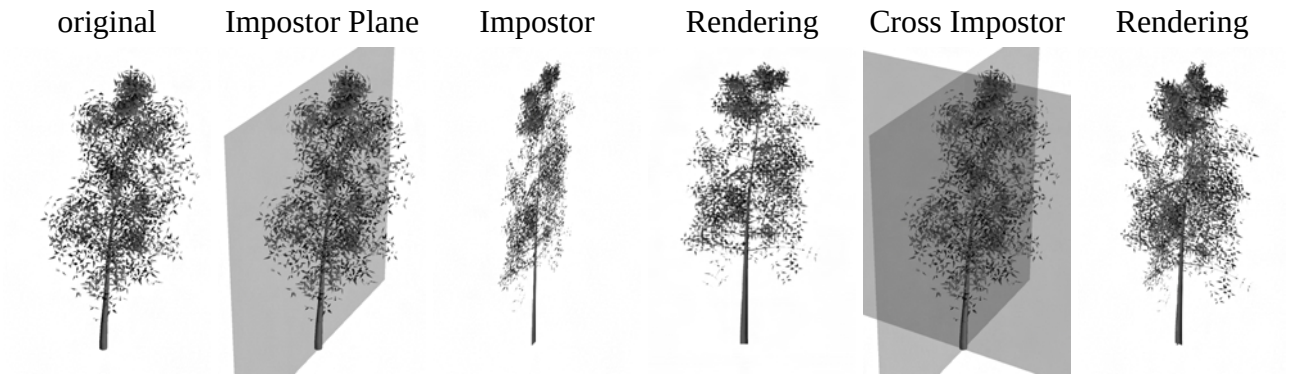



**Fig. 5.5: Impostor representations of a tree model and rendered images.**

viewer changes her position, which can only be resolved by frequently updating the texture. Therefore, following publications proposed to enrich the texture with information to recover the desired view-dependent effects. Schaufler [448] was the first to store additional depth information per pixel, which can be employed to reproduce accurate parallax effects using image warping. Shade et al. [466] stored multiple depth values, resulting in a more general scene representation. Unfortunately, forward image warping cannot efficiently be computed on graphics hardware. A similar approach was first taken by Schaufler [449]: Instead of a single impostor plane, he used multiple, parallel planes, which capture depth information explicitly. Unfortunately, these approaches do not yield representations that avoid run-time updates. Determining the circumstances under which an impostor with view-dependent geometry needs to be updated unfortunately requires carefully designed metrics which are difficult to evaluate.

Therefore, impostors with view-independent geometry were proposed. Initially, objects like trees which are approximately rotationally invariant w.r.t. some axis were approximated by crossing billboards (cf. Figure 5.5). Later, similar to remeshing techniques, Décoret et al. [97] proposed to represent general objects by sets of optimally and automatically placed planar impostors, with textures encoding colors and normals. The name *Billboard Cloud* was coined for such a representation. The authors presented a placement strategy for the impostor planes based on a greedy method operating in Hough space and achieved good results for extremely simplified objects (cf. left part of Figure 5.6). Unfortunately, their impostor placement strategy does not take into account orientation, leading to severe problems if the view direction becomes perpendicular to an impostor plane. Therefore, other plane placement techniques were proposed. Andujar et al. [6] propose to voxelize watertight models and greedily select planes that cover the largest portion of the model not handled so far. In contrast to the method of Décoret et al. they consider orientation information. Unfortunately, their approach is limited to watertight models. More suitable approaches for general meshes are proposed in Chapter 7.

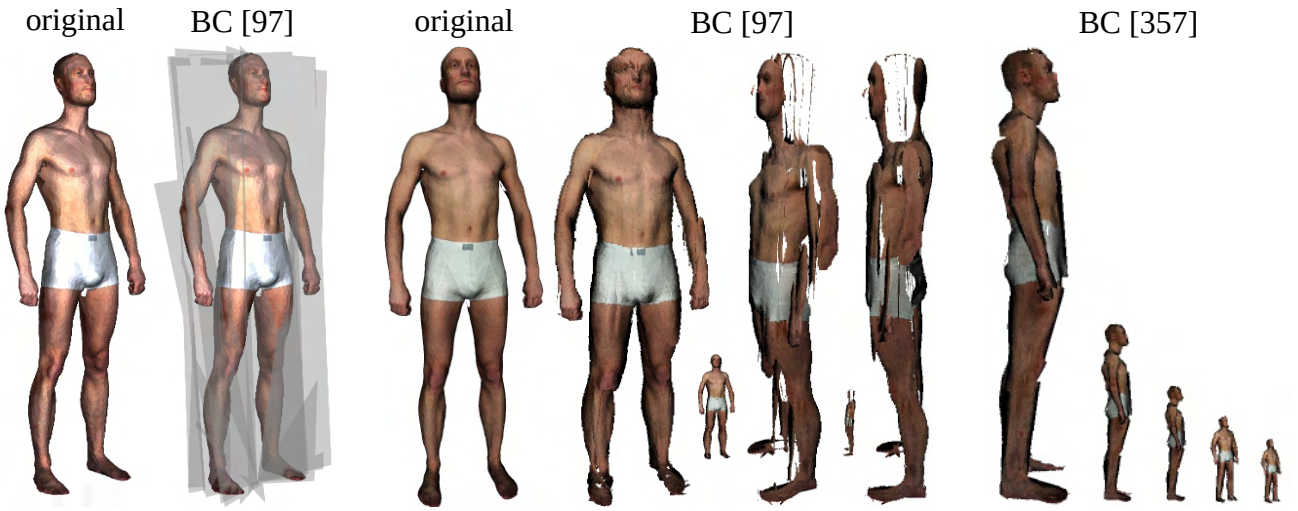



**Fig. 5.6: Billboard Clouds. Left: The original model is approximated by a set of planes. Center: If viewed from the front, especially when seen at a distance, the BC resembles the original closely. Unfortunately, most impostor planes are almost parallel, leading to problems for view directions perpendicular to the planes. These problems can be avoided using different plane placement techniques [357] (right).**

Compared to impostors with view-dependent geometry, Billboard Clouds need not be updated at run-time. This makes them suitable for use in many rendering systems since they are much easier to handle and since they avoid typical problems of IBRs like dis-occlusion artifacts [242]. If Billboard Clouds with normal maps are used, even accurate view-dependent shading can be achieved. Chapter 7 additionally introduces a method for preserving complex material properties when generating Billboard Clouds from input objects. Therefore, these representations can efficiently be used for low-resolution display of arbitrary objects. Unfortunately, the use of multiple textures leads to significant memory requirements. Therefore, Billboard Clouds are typically restricted to extremely simplified objects, since they require few low-resolution textures only.

Compared to incremental simplification methods and remeshing techniques, image-based object representations feature much lower geometric complexity and thus enable more efficient rendering. This reduction in complexity is traded for additional memory consumed by textures that store surface information like normals and material properties. Due to these storage requirements, especially impostors with view-independent geometry are most suitable for drastic simplification. Therefore, they can often efficiently be combined with incremental simplification methods to cover the full range of LODs [3, 581].

### 5.2.2 LOD Metrics for Triangle Meshes

While LOD operators specify what to do in order to simplify or refine a mesh, they do not define according to which criteria this change of complexity shall occur. Influences of complexity changes onto the accuracy of a mesh are typically computed from *error metrics*. Error metrics therefore control the application of simplification operations, e.g., in the case of iterative simplification, they define the order of application of operations.

#### Geometric Error

The error metrics proposed most early exclusively focused on assessing geometric accuracy, i.e., the geometric approximation error introduced when replacing an input mesh $M_n$ with $n$ triangles by a simplified version $M_i$ ($i < n$). Klein et al. [271] advocated to compute this simplification error from the Hausdorff distance

$$d_H(P_1, P_2) = \max_{p_1 \in P_1} \left\{ \min_{p_2 \in P_2} \{d(p_1, p_2)\} \right\}$$

of two (infinite) sets of points $P_1 = M_n$ and $P_2 = M_i$, typically assuming a Euclidean distance function $d$ for point-to-point distances. Due to the asymmetric nature of $d_H$, usually the symmetric version $d_{H_s} = \max\{d_H(P_1, P_2), d_H(P_2, P_1)\}$ is used. Unfortunately, efficient computation of $d_H$ and especially $d_{H_s}$ is very difficult. Klein et al. [271] presented a method for incremental computation of $d_H$ in combination with incremental simplification schemes. A rather efficient technique for computation of $d_{H_s}$ which concentrates efforts on regions of potentially highest distance was described by Guthe et al. [187] but remains computationally challenging. Cohen et al. [71] instead enforced an upper geometric bound on the simplified distance based on $d_H$. They check whether simplified entities lie within a volume derived from offsetting the original surface by a predefined value into either direction. A more recent, efficient implementation of this approach is based on a voxelized representation of the offset volume [608].

Other researchers proposed more efficient metrics approximating $d_{H_s}$. In a highly influential publication, Garland and Heckbert [156] introduced error quadrics as a simplification metric. They are used to measure the sum of squared distances of a point to the set of planes defined by the triangles adjacent to a vertex. In combination with vertex clustering and edge collapse operations, they efficiently determine the simplification error introduced by replacing a vertex by a different one. Since they can be specified by the ten coefficients of a symmetric $4 \times 4$ matrix, they can be stored efficiently. Additionally, summation of quadrics yields a conservative approximation of the

simplification error introduced in a series of dependent simplification operations. Therefore, they enable efficient error propagation, making it possible to impose upper error bounds on models derived from iterative simplification. This is an essential advantage over early approaches as the one of Schroeder et al. [461] which lacks error propagation. Finally, minimizing error quadrics leads to optimal positions for newly introduced vertices. Compared to error metrics based on Hausdorff distance, error quadrics can be computed much faster. Yet, a significant problem of this approach is overestimation of actual simplification error when summing error quadrics, which prohibits aggressive simplification and thus performance-optimal LOD models.

To avoid these overestimations, Lindstrom and Turk [316] proposed a simplification metric that does not require error propagations. They advocated to measure volume, surface area, and boundary preservation when simplifying meshes. Using their technique, more drastic simplifications can be computed than with error quadrics. Unfortunately, it guarantees no upper bound on the induced geometric simplification error.

### Surface Attributes

The goal of model simplification is generation of new models that are visually indistinguishable from the input object. Since many attributes of an object contribute to its visual appearance, error metrics need to take more criteria into account than geometric deviation.

Garland and Heckbert [157] were the first to propose error metrics that consider surface attributes in addition to position. They extended error quadrics such that arbitrary, linearly interpolated appearance values like colors, texture coordinates or normals could be handled. To accomplish this task, all relevant attributes are concatenated into high-dimensional vectors and an error quadric in this higher-dimensional space is defined. While this approach works fine in principle, it introduces an equal weighting among the different attributes which is neither naturally defined nor suitable for most applications. A more efficient and accurate version of these generalized error quadrics was proposed by Hoppe [223] but it features the same weighting problems. Recently, error quadrics have been further extended to preserve distinctive feature points, edges, or regions [266, 415] by introducing penalty weights for removing such features.

A metric measuring the combined error introduced by geometry and normal deviations was proposed by Guthe et al. [185]. Assuming linear interpolation of normals as in the Phong shading model [413], the authors showed how these two deviations can be combined in a meaningful way such that shading error is approximated.

While metrics proved reasonably suitable for measuring errors introduced by changes of positions, colors and normals, it is much more difficult to assess the influence of

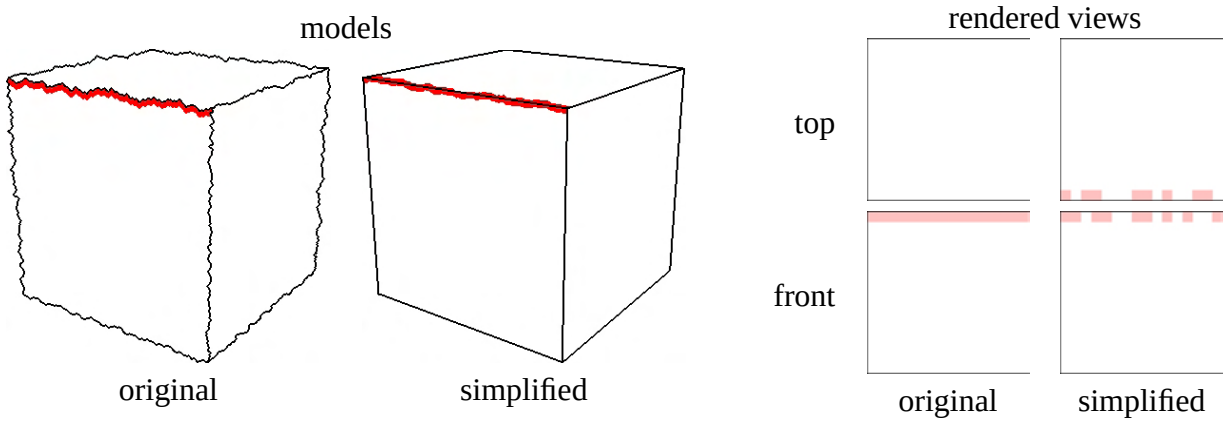



**Fig. 5.7: Simplifying textured models may have drastic influence on object appearance even if texture deviation is controlled. The original model of a jaggy box contains a single red marker stripe on the front face. After simplification parts of this stripe may appear on the top face, leading to wrong rendering results (right part, MIP-mapped texture shown).**

changing texture coordinates, which control the assignment of material to the object. Cohen et al. [70] proposed to measure texture deviation (i.e., the amount by which a texel moves at most on the simplified mesh) and to reassign texture coordinates minimizing this metric to new vertices. Although the metric suffices to control the amount of texture movement on the surface, it omits the textures content (i.e., properties like regularity or uniformity). Therefore, estimations of the amount of perceived errors due to texture deviation cannot be provided (c.f., Figure 5.7).

To actually derive an assessment of visual difference between meshes of different complexity, Lindstrom and Turk [317] proposed to measure simplification error from an image-based metric. They rendered images of two models with different complexities from various view directions and then applied image comparison metrics to assess visual similarity. Although this metric requires no weights for the various appearance attributes and although it represents the most suitable metric for estimating perceived differences publish so far, it is rarely used due to the large computational overhead for rendering and comparing large numbers of images.

To avoid visual degradation due to simplification of appearance attributes, image-based object representations resample surface properties. For iteratively simplified meshes such an approach was proposed by Cignoni et al. [67]. They advocate resampling of the properties into textures that are assigned to the simplified mesh. After creating a simplified mesh and parameterizing it, textures are created by filling them with appearance properties from the original mesh, i.e., by recording properties from the point on the original mesh closest to the point on the simplified mesh where the respective texel is located. To avoid ambiguities when determining closest points on the

**Fig. 5.8: Static LODs of a VW Golf car body. Left: original (43k vertices), center: 3 mm geometric error (7k vertices), right: 1 cm (6k vertices).**

original mesh, techniques like the one of Collins and Hilton [75] can be used, which guarantee existence of an injective mapping. Unfortunately, the technique of Collins and Hilton is limited to manifold meshes. Therefore, the approach of Cignoni et al. will lead to problems for general meshes.

Compared to other representations, the huge advantage of approaches like image-based object representations that resample surface attributes is that they need to take into account geometric error only. All other surface properties can accurately be represented as textures at the cost of additional memory requirements.

### 5.2.3 LOD Datastructures for Meshes

In the previous section criteria and methods for simplification of triangle meshes were presented. This section now presents data structures that enable efficient access to LOD representations and thus efficient changes of LOD. Such data structures are extremely important not only for run-time selection of LODs but additionally enable efficient off-fline generation of LODs.

#### Static LODs

The most simple LOD representation are sets of *static LODs*, i.e., sequences of simplified versions of triangle meshes where the versions were simplified up to some target simplification errors. Figure 5.8 shows a sequence of three such versions of a car body mesh.

Typically such static LODs are managed by scene graphs which activate the most suitable representation at run-time depending on some selection criterion (see following section). Special care has to be taken to avoid *popping artifacts*, i.e., sudden changes of the object's appearance when switching the displayed LOD.

Static LODs are commonly derived from iterative mesh simplification algorithms, remeshing techniques, and approaches creating image-based representations. Mixing

various LOD representations like simplified meshes and Billboard Clouds is easily possible since separate LODs are handled independently of each other. They are used in most real-time graphics applications showing complex environments like VR or games.

### Progressive LODs

Whereas static LODs are very simple to handle and quite efficient in terms of storage, they feature a significant disadvantage: The resolution of the displayed model is fixed. As a result, if some parts of the model need to be displayed at a high resolution, the whole object has to be rendered at this high resolution. In situations where the extent of the displayed object is relatively large, this increases the number of displayed triangles significantly. To solve this problem, the idea of *dynamic LODs* was developed. Such LOD representations enable much finer control over the resolution of displayed objects, effectively allowing varying resolutions for separate parts of a simplified mesh.

A widely known approach to dynamic LOD are *progressive meshes*, which are derived by iteratively simplifying an input mesh and storing the sequence of simplification operations and the simplified mesh $M_0$ in an appropriate data structure like the Multi-Triangulation data structure [425]. Meshes of arbitrary resolution can now be derived from the stored information by selectively refining $M_0$ by applying inverse simplification operations. To generate LODs where the accuracy varies over the model, the refinement operations need to be applied in a different order than the inverse stored sequence. This is possible since most simplification operations are independent of each other and thus the sequence defines a partial order only.

Efficiently specifying and storing this partial order unfortunately is not trivial. Several researchers [598, 222, 126, 267] provided definitions for the dependencies between simplification operations, resulting in more or less restrictive partial orders, i.e., allowing less or more aggressive simplification or refinement in specific regions of the object. Unfortunately, not all of these definitions lead to meshes that avoid foldovers.

Dependencies can be minimized by choosing appropriate simplification sequences. Xia and Varshney [598] proposed to simplify a mesh in several iterations. In each iteration, only those simplification operations are performed that are independent of any other operation executed in the same iteration. Diaz-Gutierrez et al. [104] optimized this scheme by solving a corresponding weighted perfect matching problem.

Progressive meshes are derived from iteratively simplifying meshes. A different progressive representation, i.e., subdivision meshes, can be derived using surface fitting and remeshing techniques. Progressive LODs are usually employed in applications like terrain visualization, where the displayed model is large compared to the viewer, allowing for largely varying degrees of detail over the model.

### Hierarchical LODs

While the use of static LODs is relatively unflexible compared to progressive meshes, they require much less run-time overhead. At each frame, only the most suitable LOD version has to be chosen. In contrast, progressive meshes need to readjust the model complexity each frame, which implies validity checks for all active vertices. For large models and incoherent animation sequences this overhead easily outweighs the advantages of rendering less triangles.

Therefore, Erikson et al. [128] proposed *Hierarchical LODs* (HLODs). To create them, a scene is subdivided into separate parts in a hierarchical way (e.g., using a scene graph [128, 531], hierarchical face clustering [604], or spatial subdivision data structures [315, 43, 468]). Then, the partitioned geometry parts are simplified individually. The HLOD concept leads to three significant advantages: First, they allow for dynamic LOD by choosing separate resolutions for different parts of the geometry. Second, compared to progressive meshes, selection of LOD levels becomes much more efficient since mesh resolutions are selected for chunks of geometry instead of on a per-vertex basis. Third, hierarchical subdivision of the input model naturally lends itself to out-of-core handling, i.e., processing of data sets too large to fit into main memory.

Obviously, HLODs also introduce specific problems. If geometry parts are simplified individually, which is especially relevant for out-of-core models, gaps may be introduced into simplified models since simplifications in neighboring mesh parts are not taken into account. To solve these problems, either mesh elements associated with multiple mesh parts are left unchanged, which leads to bad simplification rates, or the separate pieces are merged using tedious stitching operations [424, 43]. In combination with careful control of the geometric simplification error and techniques to close gaps during rendering [18] the stitching can be avoided [186]. Cracks can be avoided completely at the cost of introducing further dependencies between mesh parts by either enforcing the same simplification order as for progressive meshes [125], which leads to frequent loading and unloading of geometry parts for out-of-core models, or by loading multiple parts at once [68], which introduces disc access overheads and increased memory requirements.

HLODs are typically created using iterative simplification methods, but remeshing techniques could be employed as well. Gobbetti and Marton also proposed a technique based on HLODs containing image-based object representations [163]. Due to the good tradeoff between rendering efficiency and management overhead, HLODs are employed in a large number of applications today, ranging from terrain visualization over rendering applications for archaeological data to out-of-core data visualization.

### Out-of-Core LOD Datastructures

Specific data structures are necessary for out-of-core handling of huge models. They are responsibility for providing efficient access to all necessary data while reducing the memory footprint as much as possible. Typically, this is achieved by subdividing the geometric model in a hierarchical way (either using HLODs or chunks of progressive mesh elements [98, 604]), storing these parts in files, and keeping an index table for these parts in main memory. At run-time, chunks are loaded dynamically from disk. Obviously, for optimal performance, it becomes inevitable to minimize latency times by choosing optimal data layouts and access patterns [590, 232, 603], and by applying data compression techniques [170] to reduce data transfer times.

## 5.2.4 Run-Time Management of LOD Triangle Meshes

Having introduced various LOD datastructures in the previous subsection, this part will describe how the appropriate LOD versions can efficiently be chosen and rendered at run-time.

### Selection Metrics

In the same way as simplification is driven by simplification metrics, LOD selection is guided by selection metrics. They approximate the accuracy that a LOD model needs to have to guarantee satisfying display of the model. Aiming at predictive rendering, this obviously implies selection of models such that rendered images feature radiometric correctness.

Practical selection metrics typically consider just very simple properties. The most simple ones select LODs based on the distance to the object or its projection size, which enables approximations of geometric errors. More accurate metrics [597, 270] demand higher detail in regions corresponding to silhouettes and those showing shading highlights. A metric also taking into account texture deviation was proposed by Schilling and Klein [452]: It approximates the view-dependent color changes due to textures, also taking into account texture properties (e.g., uniformity). Another texture-aware metric was proposed by Williams et al. [580], who extended the simplification metric of Cohen et al. [70]. In addition to texture deviation, it captures contrast sensitivity, which has a strong influence on human perception. A specialized metric for LOD meshes casting soft shadows onto other parts in the scene was proposed by Guthe et al. [185].

### Efficient LOD Rendering

Efficient rendering of selected LODs needs to take into account the properties of current graphics hardware. Large improvements of rendering speed can be achieved on existing GPUs by employing most optimal rendering methods [284] (e.g., vertex arrays) and by rendering triangle strips instead of individual triangles. Numerous approaches for triangle strip generation exist [599, 224, 127] which vary largely in the time requirements for computing triangle strips and the optimality of generated results. While triangle strips of static LODs and HLODs can be computed in preprocessing steps, such an approach is not possible for progressive meshes. Therefore, on-line triangle strip management schemes have been proposed [124, 467].

In addition to this, huge performance improvements can be achieved in combination with culling techniques, which remove mesh elements that do not contribute to the final image. For out-of-core rendering, significant speed-ups are often derived from multi-threaded rendering systems and suitable data-prefetching schemes.

## 5.2.5 Discussion

Summarizing the previous sections, existing LOD techniques enable production of high-quality renderings from huge polygonal models at interactive frame rates. They do not only preserve the geometric properties of an object but additionally several appearance attributes like normals and colors. Therefore, they are very suitable for increasing rendering speed and reducing aliasing problems.

In contrast to preservation of normals and colors, preservation of surface materials is still a problem. Published approaches controlling the texture deviation minimize the amount of texture movement on the surface. Often, this suffices, but no guarantees on the perceived difference of the original and the simplified model can be derived. Image-based object representations circumvent this problem by resampling surface attributes into textures at the cost of requiring significant memory requirements. Especially problematic is that a clear separation of material and geometry becomes unsustainable since larger-scale geometry is encoded in image-based representation. This increases the demands on respective data compression techniques.

Currently, image-based object representations are only suitable for representing distant objects. Chapter 7 presents an approach for extremely simplified objects which preserves spatially varying materials on the object by resampling them into textures which are compressed for efficient handling. In the future, extensions of this approach might make general appearance preserving LODs for predictive rendering possible.

In addition to the material appearance preservation problem, LOD representations

for meshes currently have problems with representing deformable objects. Existing publications targeting this problem introduced time-dependent LOD data structures for meshes [470, 471], which can be applied for time-dependent LODs of predefined animation sequences [366, 268]. Yet, arbitrary deformations not known in advance are currently supported for skeletal animations only [99]. Although many VR applications just require static models, many of them would profit from deformable objects. In the future, it therefore seems very likely that more research will be conducted into this direction.

## 5.3 Discussion

Predictive Virtual Reality is required at various stages in the product development process. Today, Virtual Prototyping is mainly restricted to verification of object shape, which allows for assessment of functionality and security aspects. In the future, other aspects like product design will be supported, since they allow for efficient ways of customizing products to the purchasing patterns of specific groups of customers. Therefore, since a unified solutions seems out of reach, it appears suitable to develop various kinds of predictive rendering software which support the individual development steps in an optimal way. Specifically, this requires specialized tools for accurate visualization of shape, and others for predictive rendering of products including surface materials, which are required for design reviews.

In the following two chapters own contributions to both approaches are presented. Chapter 6 presents an efficient approach for LOD management of models composed of high-level surfaces (e.g., trimmed NURBS surfaces). It might help to replace current approaches for rendering of trimmed NURBS surfaces by on-the-fly tessellation algorithms since it reduces run-time overheads and memory requirements significantly. This is especially important for highly complex models like air-planes, ships, or cars.

In Chapter 7 an image-based object representation for triangle meshes with view-independent geometry is presented, which preserves the appearance of objects covered with complex surface materials like BTFs. Like other IBR representations, it is best suited for representing distant objects, although it features less memory requirements than comparable approaches.

# Chapter 6

# Seam Graph

Interactive visualization of CAD data is of great importance for industrial VR users. Unfortunately, as the previous chapter showed, the most commonly employed CAD surface type, the trimmed Non-Uniform Rational B-Spline (NURBS) surface, cannot be rendered efficiently on existing graphics hardware. Therefore, current techniques for fast visualization typically rely on pre-tessellated models. The drawback of this approach is that surface inaccuracies become visible if surfaces are rendered above a maximum display resolution since the accurate trimmed NURBS representation is discarded.

Therefore, approaches relying on on-the-fly tessellation were proposed. As shown in the previous chapter, these techniques lead to significant reductions of memory requirements and are able to render surfaces at almost arbitrary precision. Nevertheless, they feature processing overheads due to time-consuming on-line tessellation. These overheads can be minimized if the number of surfaces which require retessellation is minimized. This chapter proposes such a technique based on a Level of Detail (LOD) representation of the trimmed NURBS model.

The idea of the approach is illustrated in Figure 6.1: The NURBS object is represented by a graph of seams, i.e., common trimming curves of neighboring patches. Simplifying this graph reduces the complexity of the object since it reduces the number of NURBS surfaces that model the object. At run-time this *Seam Graph* aids at determining those surface patches which need to be displayed. In addition to reducing tessellation overheads, the Seam Graph eliminates cracks between trimmed NURBS surfaces since it enforces common vertices along the shared trim curves.

In the following, the necessary steps for use of the Seam Graph are detailed. First, setup of the Seam Graph including respective data types is described. Then, the LOD operations on the Seam Graph are presented, followed by a description of online selection of suitable LODs. Next, an interactive rendering technique for NURBS models

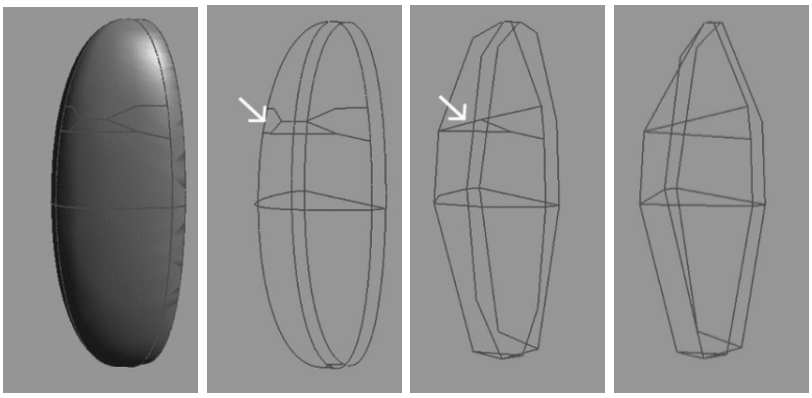


**Fig. 6.1: Seam Graph of a simple NURBS object showing vanishing patches.**

using the Seam Graph is overviewed. The final two sections give details on the achieved results, draw conclusions, and sketch opportunities for future improvements.

## 6.1 Seam Graph Setup

The Seam Graph is a data structure fulfilling two different purposes. First, it is supposed to minimize the number of rendered NURBS patches, which is especially suitable in combination with compute-intense on-the-fly tessellation methods. Second, the information stored in the Seam Graph leads to tessellated LOD NURBS models without cracks between adjacent NURBS surfaces, which increases rendering quality significantly.

The individual data types of the Seam Graph data structure, which was designed to handle non-manifold surfaces, and their relationships are shown in Figure 6.2. All of the types correspond to geometric entities. The *Patch* data type represents a trimmed NURBS patch and stores a list of *Boundaries*, which correspond to the trimming curves of the patch. For sake of simplicity, trimming curves are limited to line strips. Every Boundary consists of one or several *Boundary Segment* instances which represent non-self-intersecting, non-self-touching parts of the Boundaries. Every Boundary Segment consists of a list of *Boundary Vertex* instances, which are the most essential data type for creating crack-free tessellations. They correspond to vertices in 3D Euclidean space that are shared among the trimming curves of adjacent patches. Enforcement of these common vertices closes gaps between adjacent NURBS patches. Thus, these vertices are called *sewing points* in the following. Pairs of sewing points that are listed consecutively in this list form an edge of the trimming poly-line. Every element of the list can either be active or not, meaning that the referenced vertex is currently part of the trimming poly-line or not. This state is used during the simplification and LOD selection

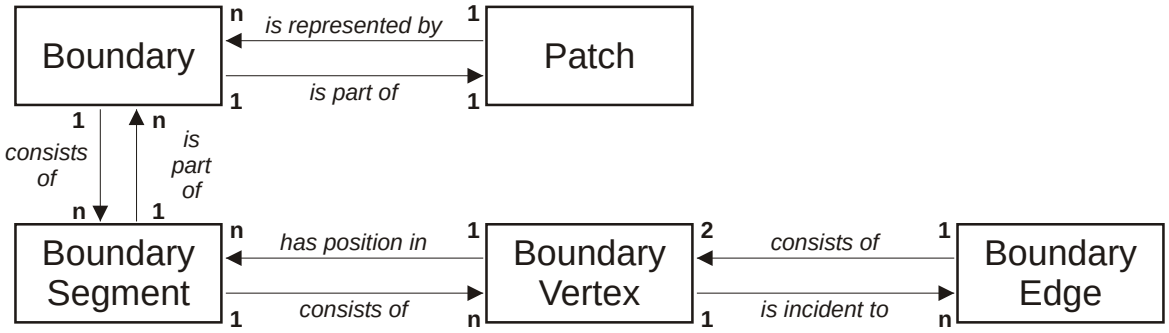



**Fig. 6.2: Relationship of Seam Graph data types.**

of the trimming curves, which is described in detail in the next section. For efficiency reasons, each Boundary Vertex stores references to the *Boundary Edges* it is part of. Boundary Vertices themselves hold references to the limiting Boundary Vertices.

An example of a Seam Graph for a small NURBS model is shown in Figure 6.1. The displayed elements are the Boundary Edges. Simplifying the graph reduces the complexity of trimming curves and the number of surfaces required to represent the original surface.

In order to create a Seam Graph for an object, connectivity information for the trimmed NURBS patches is required. Typically, such information is stored by CAD modeling systems. Unfortunately, it is seldomly exported. Therefore, it is often necessary to recompute such data [248, 189].

Setting up the Seam Graph requires the following three steps, which are illustrated in Figure 6.3. First, poly-line approximations of trimming curves are computed, yielding the Boundary entities. Special care has to be taken to limit the Euclidean geometric approximation error, e.g., employing the approach of Kahlesz et al. [248].

Second, the poly-lines are divided into segments which are shared by a fix set of patches each, yielding the Boundary Segment instances. This step is visualized in the bottom left and center images of Figure 6.3. Colored vertices belong to single patches, gray ones are shared by at least two patches. In order to generate Boundary Vertices, which are shared among all patches adjacent to a poly-line trimming curve, more vertices need to be inserted. Unshared points are projected onto corresponding trim curve segments of adjacent patches, and the projections are inserted as new points. This operation derives consistent tuples of vertices containing one vertex for each patch adjacent to the trimming curve.

In a third step, common trimming curve points are enforced in these segments by computing average positions of corresponding points, yielding the Boundary Vertex and Boundary Edge instances.

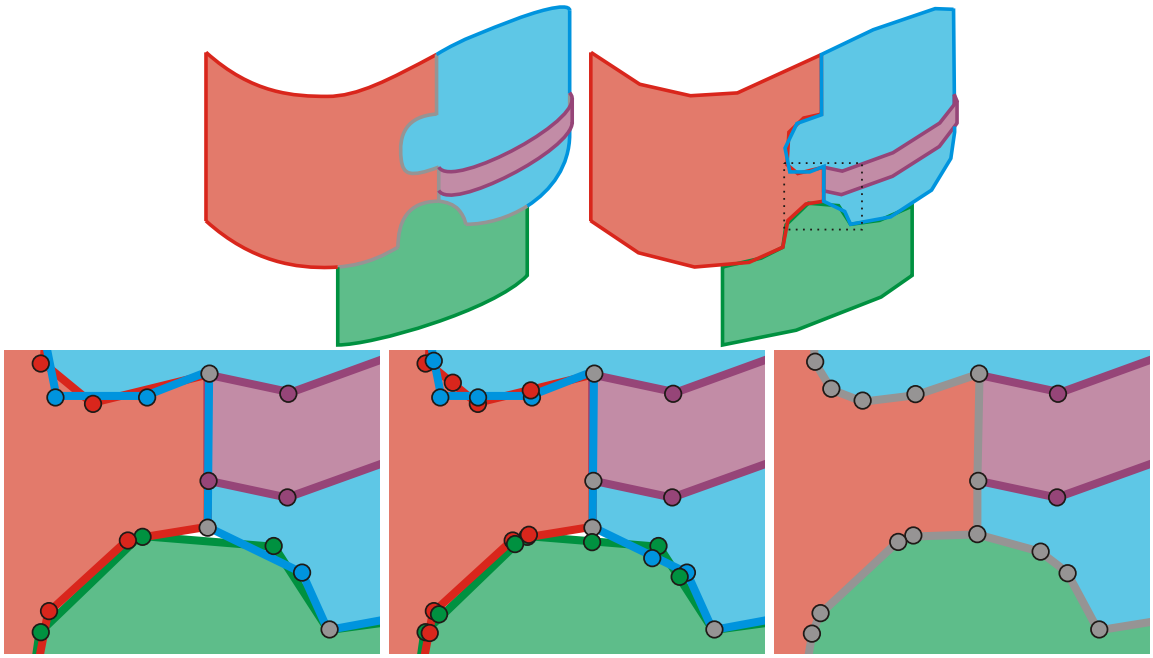


**Fig. 6.3: Seam Graph setup. Top: The trimming curves of a hypothetical model consisting of four NURBS surfaces (left) are approximated by poly-lines (right). Colored curves belong to one patch only, gray ones are shared by at least two patches. Bottom: Closeup showing the approximating polylines (left). Center: Polyline points are inserted to derive consistent tuples of vertices. Right: Boundary Vertex instances are computed from the average positions of the consistent pairs.**

## 6.2 Level of Detail

As in most mesh simplifiers that are currently used, the basic simplification operation performed on the Seam Graph is the edge-collapse operation, since it yields very accurate results and since its inverse operation – the vertex split – can easily be performed and requires few information to be stored. The basic idea of this operation is shown in Figure 6.4: Vertices $v_1$ and $v_2$ are collapsed into vertex $v'$, removing two triangles, three edges and one vertex from the original mesh. Since the Seam Graph represents no triangulated mesh, the Seam Graph edge collapse has to handle general polygons, and non-manifold edges and vertices.

Implementing the edge-collapse operation always requires two elementary functions:

- A cost function that assigns each edge a cost, thereby introducing an order among the edges reflecting the desire to collapse some edges quickly, others at a later point of time. As shown in the previous chapter, this measure typically takes into account the geometric error, texture stretch and deviation, or deviation of scalar attributes like color.

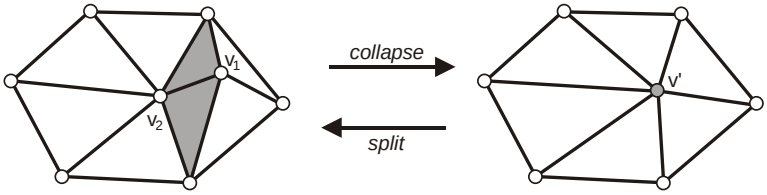



**Fig. 6.4: Edge-collapse operation.**

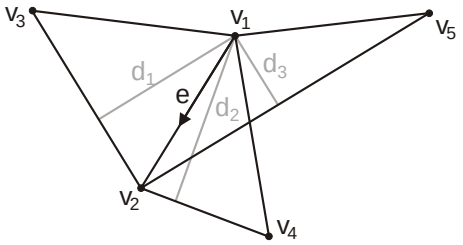



**Fig. 6.5: Error metric.**

- A placement function that decides where the vertex resulting from the edge-collapse should be positioned. Typically one tries to find a position such that the simplification metric is minimized. For efficiency reasons, it is often more suitable to restrict the simplification to a half-edge collapse operation. This operation chooses the position of the collapsed vertex from the positions of the vertices adjacent to the collapsed edge. It is employed in the following due to the following reason: Choosing new positions requires determination of respective parameter space values (e.g., by linear interpolation of existing parameter space values). Since linearly interpolated values are not guaranteed to lie on the trimming curves any more, the normals of adjacent surfaces – computed from the parameter space values – could vary significantly, resulting in visual artifacts.

## Cost Function

Since the main target of the Seam Graph are models created with CAD systems which typically have neither texture information nor color, the cost function is chosen to measure the geometric error introduced when removing an edge. As mentioned in the previous chapter, this requires computing the symmetric Hausdorff distance $d_{H_s}$, which measures the distance between two (infinite) sets of points. Unfortunately, computing it is very time consuming even if approximating sampling strategies are employed.

Another approach which would provide very tight error bounds is described by Klein et al. [271], who compute the directed Hausdorff distance $d_H$ between the original and the simplified mesh. Since computing the collapse-costs always leads to redundant computations which are unavoidable (one needs to compute for every vertex the costs for every adjacent edge, but only the minimum of these costs is assigned to the vertex), their method would increase simplification times too much, since every of their computations is already relatively slow. In order to still compute tight error bounds for the resulting meshes, the cost computation is split into two different parts. Together, they provide a close, upper bound of the real cost which can be computed much faster:

- After each edge-collapse operation, for every vertex in the 1-ring of the simplified edge the actual distance $d_H$ between the original and the current Seam Graph is computed as described in [271].

- For each edge $e$ that requires update of its collapse cost (e.g., all edges adjacent to $v'$ in Figure 6.4), the error $\varepsilon$ is computed that would be introduced to the current mesh by collapsing $e$.

The combination of $d_H$ and $\varepsilon$ provides an upper bound for the geometric error between the original mesh and the mesh after the simplification step. Since it provides an overestimation, edges might be chosen in a slightly wrong order.

In order to take $d_H$ into account for the computation of the costs for edges, for every vertex in the 1-ring of the simplified edge the maximum distance that was computed for any edge adjacent to it is stored. If such a vertex stored a maximum error before, the larger one of the existing and the new maximum error are chosen.

The error measure for $\varepsilon$ is described by the example in Figure 6.5. In this case, where $e$ is collapsed into $v_2$, the distances $d_1$, $d_2$ and $d_3$ are computed as the minimum distances between $v_1$ and the edges $(v_2, v_3)$, $(v_2, v_4)$ and $(v_2, v_5)$ respectively. The maximum of these distances describes the geometric error when collapsing $e$ into $v_2$ (this directed collapse is denoted by $v_1 \rightarrow v_2$ to distinguish it from the opposite operation $v_2 \rightarrow v_1$, where $e$ is collapsed into $v_1$).

To finally decide into which vertex the edge $e$ should be collapsed, $d_H$ and $\varepsilon$ are combined in the following way: First, the cost of $v_1 \rightarrow v_2$ is calculated and the maximum error that was stored with $v_1$ (to account for previous modifications of the mesh) is added. Then the cost of $v_2 \rightarrow v_1$ is computed and the maximum error stored with $v_2$ is added. The minimum of these costs and the according direction of the collapse are assigned to $e$.

Unfortunately, the above approach leads to two potential problems: First, since $d_H$ is not symmetric, it might lead to bad results in special cases but the conducted experiments proved it a reasonable choice for all tested models. Second, the geometric error of the trimming curves not necessarily provides an upper bound of the geometric error of the patch interior. For models that feature these problems, one could compare the bounding box of the trimming curve to be simplified against the bounding box of the associated patch and increase the collapse cost substantially if their extents vary significantly. Since the models tested in the conducted experiments never lead to any such problems since they consist of NURBS surfaces of low degree, this test was omitted in the presented implementation.

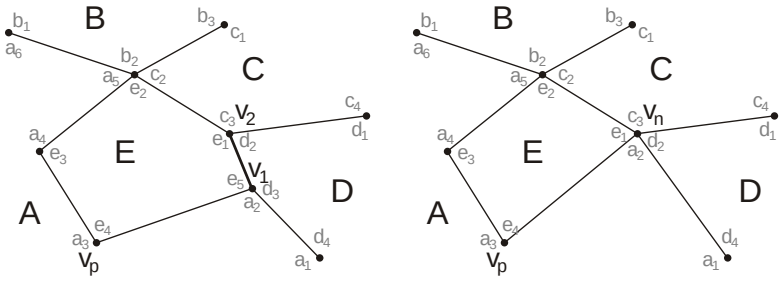



**Fig. 6.6: A half-edge-collapse operation in the Seam Graph.**

## Simplification Step

Before the first simplification step can take place, the costs for all edges are computed as described above. Then, they are stored in a heap sorted on the edges' collapse costs. The edge $e$ with lowest cost and all edges from its neighborhood are removed from the heap (here, the neighborhood of an edge is defined as those edges that are incident in any vertex of the 1-ring of one of the vertices of $e$). Figure 6.6 visualizes the following steps (note: upper case letters denote patches, vertices of special interest are assigned names starting with $v$, the other character-number combinations describe positions in the Boundary Segments' vertex lists, e.g., $e_4$ means that $v_p$ is part of the Boundary Segment of $E$ and occupies position 4 in its vertex list). In the example, $(v_1, v_2)$ is supposed to be collapsed into vertex $v_2$. A new vertex $v_n$ is created as an exact copy of $v_2$, and $v_2$ is replaced with the new vertex. $v_1$ is removed from the mesh, all edges $(v_1, v_i)$ are redirected to $v_n$ if $v_2$ was not connected to $v_i$ already, and $v_1$ is deactivated in all Boundary Segment instances it was part of. Since $v_n$ gets linked to vertex $v_p$ now, it becomes part of the Boundary Segment of patch $A$, where it is placed in position 2 - the one previously occupied by $v_1$. Such a simplification step is denoted by $(v_1, v_2 \rightarrow v_n)$.

After this, the error that the step introduced is computed as the directed Hausdorff distance $d_H$ between the original graph and all edges incident in $v_2$ as mentioned above. Thus the current geometric error is given by the maximum of the previous geometric error and the error of the current step.

After penalizing $v_n$ and the vertices adjacent to it, the collapse costs for edges in the neighborhood of $v_n$ are updated (here, neighborhood describes all edges that are incident in vertices adjacent to $v_n$). These costs are reinserted into the cost heap and the single simplification step is finished. Please note that a subsequent simplification of edge $(v_n, v_p)$ would remove patch $E$ from the Seam Graph. This way, tessellations of simplified models with less triangles than the number of patches in the original model become possible.

The simplification algorithm stops, if no further edges can be collapsed, which is the

case when just a single vertex per connected part of the model remains.

For progressive representation, the required edge-collapse information is stored. To reconstruct an edge collapse, only the indices of the vanishing vertices and the new vertex need to be stored. Additionally, for later LOD adjustments the current geometric error is saved.

### Adjusting the view-dependent Level of Detail

Selection of an appropriate LOD for the Seam Graph requires implementation of two operations:

- A refinement operation that takes as input the collapse information for a simplification step $(v_1, v_2 \rightarrow v_n)$ and performs a vertex-split $(v_n \rightarrow v_1, v_2)$. The split operation replaces $v_n$ by $v_2$ and reinserts $v_1$ at its original position. Additionally, all the edges that originally were incident to $v_1$ and $v_2$ are redirected to $v_1$ respectively $v_2$, or reinserted into the mesh if they were removed before. The result of the operation is the original configuration as before the step took place.

- The simplification operation as described above. Obviously, once the progressive LOD hierarchy is created, it needs not store any more information or generate new vertices.

As described in the previous chapter, simplification steps depend on each other. The most efficient scheme for coding these dependencies was proposed by El-Sana et al. [126]. It works by assigning each vertex of the mesh a number: Vertices of the original mesh with $m$ vertices are assigned distinct natural numbers from $1$ to $m$, vertices created by a simplification step are numbered in order of creation $(m+1, m+2, \ldots)$. For each vertex $v$ that participated in an edge collapse operation (either $(v, v_2 \rightarrow v_n)$ or $(v_1, v \rightarrow v_n)$ ), the number of its child is stored (note: in case of a simplification step $(v_1, v_2 \rightarrow v_n)$, we call $v_n$ the child of $v_1$ and $v_2$). Given this scheme, an edge collapse $(v_1, v_2 \rightarrow v_n)$ can be performed if all vertices adjacent to either $v_1$ or $v_2$ have numbers smaller or equal to $m$ (and thus represent vertices from the original mesh) or if the numbers of the children of the adjacent vertices are smaller than the number of $v_n$. A vertex split $(v_n \rightarrow v_1, v_2)$ can be performed if all vertices adjacent to $v_n$ are assigned a number smaller than that of $v_n$. In cases where this split should be performed but is not possible, vertices adjacent to $v_n$ with bigger numbers than $v_n$ need to be split.

For view-dependent rendering, the LOD of the displayed object has to be adjusted every frame. Clearly, always refining the coarsest Seam Graph version up to the desired accuracy is very inefficient. A solution to this problem employing temporal coherence

was proposed by Xia and Varshney [598]: Only for the first frame the set of active vertices is derived from the coarsest resolution Seam Graph. In following frames, the set of active vertices from the previous frame is updated to match the resolution of the current frame by applying suitable simplification and refinement operations. Whenever an edge should be collapsed but dependencies prohibit this operation, the edge is left unchanged. In case a vertex split is prohibited by dependencies, all adjacent vertices disallowing the operation are recursively forced to be split until the intended operation can be performed. Managing active vertices leads to significantly increased performance since the desired LODs do not change drastically from frame to frame.

## 6.3 Rendering NURBS Models

The Seam Graph is designed to improve the rendering speed of on-the-fly tessellation approaches for trimmed NURBS surfaces. Yet, no assumptions about the specific rendering technique are made as long as common sewing points can be enforced. In the following, an approach employing standard, CPU based tessellation of individual patches is described. Alternatively, more recently developed and more efficient rendering methods like the GPU-based rendering technique of Guthe et al. [184] could be used.

Adaptive tessellation methods [189, 17] guaranteeing a certain geometric approximation error are rather costly. Therefore, the number of patches to be tessellated per frame needs to be minimized using the LOD from the Seam Graph and per-patch culling techniques. In addition, load balancing schemes can restrict the number of newly tessellated patches per frame. Note that in combination with on-the-fly tessellation methods, the management costs for the Seam Graph are negligible. Yet, this overhead becomes significant for exceptionally complex models. In such cases load balancing schemes need to be devised for Seam Graph management as well, but none of the models tested made this necessary.

### LOD selection

Prior to selecting the view-dependent LOD, an upper bound $\varepsilon$ for the geometric error needs to be computed per patch. Therefore, first the point $p$ on the patch's bounding box which is closest to the eye-point $\nu$ is determined. If the distance $d = \|p - \nu\|$ is smaller than the length of the diagonal of the patch's bounding box, a better approximation is calculated using the distance to the nearest vertex of the tessellated patch.

Next, the error $\varepsilon$ is computed such that an edge of length $\varepsilon$ at distance $d$ from $\nu$ projects to at most half a pixel on the screen. Additionally, the calculated error is required to be no less than ten percent of the sewing error to restrict the maximum triangles per

patch. These error bounds are passed to the Seam Graph which selects the LOD for the trimming curves in such a way, that the geometric error assigned to the trimming curves is always smaller than the acceptable geometric errors of the adjacent patches.

### Culling

Since the Seam Graph algorithm keeps the patches as separate objects, per-patch culling techniques can easily be employed, reducing the number of retessellated patches and the number of triangles sent to the graphics pipeline. All kinds of culling mechanisms (i.e., view-frustum, backface, and occlusion culling) can be combined with the rendering algorithm. In the following, existing implementations of view-frustum and backface culling are described. Occlusion culling can efficiently be integrated on the basis of the technique of Bittner et al. [32].

The view frustum culling approach is straightforward and works by testing the bounding box of each patch against the current view frustum, which is defined by the four sides of a semi-infinite pyramid. The bounding box of a patch lies inside the view-frustum if all eight corners have positive distances to all four planes, given a consistent orientation of the planes (i.e., with plane-normals pointing towards the inside of the pyramid).

The back-face culling algorithm is based on the normal-cone approach [479]. For each patch, a cone is computed such that all normals of the patch lie within this cone. In the described implementation, such a cone is determined per patch using the normals of the vertices of the finest possible tessellation, which is calculated in a preprocessing step. In order to conservatively determine the visibility of the whole patch, a visibility test (in terms of backface-culling) is performed for the normal cone and every corner of the bounding box. If all tests determine invisibility, the whole patch is facing backwards and thus can be culled.

### Load Balancing

Load balancing tries to equalize frame rates by equally distributing work over several frames. In the context of on-the-fly tessellation of NURBS surfaces, load balancing needs to equalize the number of patches retessellated per frame. During adjustment of the LOD in the Seam Graph a vertex split or edge collapse is only allowed if the total number of updated patches is below a given maximum. To concentrate tessellation efforts on patches $S_i$ where the current and the target approximation errors $\varepsilon_{i,c}$ and $\varepsilon_{i,t}$ deviate most, a priority scheme is introduced. For every patch a priority $p_i = \max\{\varepsilon_{i,c}/\varepsilon_{i,t}, \varepsilon_{i,t}/\varepsilon_{i,c}\}$ is introduced, and only patches with highest $p_i$ are chosen for tessellation. To avoid retessellation of invisible patches, which should be handled

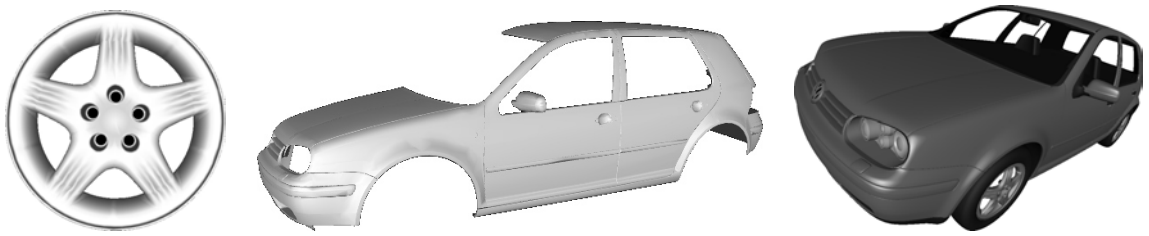


**Fig. 6.7: Tested NURBS models. Wheel rims (one of two shown), car body, and assembled car.**

only after adjusting errors of visible patches, the priority scheme is modified as follows:

$$p' = \begin{cases} p & \text{patch not culled} \\ 1 - 1/p & \text{patch culled} \end{cases}.$$

Tessellation is stopped if the new patch would increase the number of tessellated triangles to more than a given maximum or all patches are retriangulated.

## 6.4 Results

The LOD rendering algorithm was tested with three trimmed NURBS models on a 1.8 GHz Pentium 4 with 512 MB memory and an nVIDIA Geforce 3 MX graphics board. The tessellation algorithm could handle 25k parameter space triangles per second, and about 250k parameter space triangles could be updated with new 3D coordinates. Given a target frame rate of 25 fps, the abovementioned update restrictions were set to 1k new and 10k updated parameter space triangles per frame. If the visible error $\varepsilon_{vis}$ exceeds one pixel, the number of new triangles is modified to be 1k$\cdot\varepsilon_{vis}$, reducing the error at the cost of lower frame rates.

Table 6.1 gives an overview of the computation results of the three models (cf. Figure 6.7). The first model consists of the two wheel rims from the half car, whereas the second model consists of the body from the half car. The third model is the assembled complete car including lights, wheels and plastic parts. In preprocessing steps, the connectivity information was computed, and the Seam Graph was set up employing a trimming curve approximation error of 0.2 mm, resulting in a maximum LOD of 0.02 mm for the triangulation.

The combination of LOD selection, culling, and view-dependent tessellation leads to displayed models with relatively small numbers of triangle, even if the input model is rather complex. As Table 6.1 shows, the maximum number of displayed triangles is only about 1.5 to 5 percent of the number of triangles resulting from tessellating the whole model with the target approximation error of 0.02 mm. The frame rates are interactive

| | wheel rims | car body | full car |
|---|---|---|---|
| NURBS patches | 302 | 1.6k | 8.0k |
| Bézier patches | 3.7k | 1.8k | 17.7k |
| triangles (0.02 mm) | 380k | 637k | 3.6M |
| memory (MB) | 15.6 | 27.3 | 151.9 |
| bound. edges | 23k | 56k | 278k |
| max. triangles | 18k | 11k | 50k |
| min. fps | 9.9 | 4.2 | 2.1 |
| avg. fps | 29.7 | 15.8 | 8.3 |
| max. error (pixel) | 1.6 | 1.7 | 3.7 |
| memory (MB) | 8.1 | 20.8 | 103.1 |

**Tab. 6.1: Overview of rendering results on a 1.8 GHz Pentium 4 (512MB memory) and nVIDIA GeForce 3 MX.**

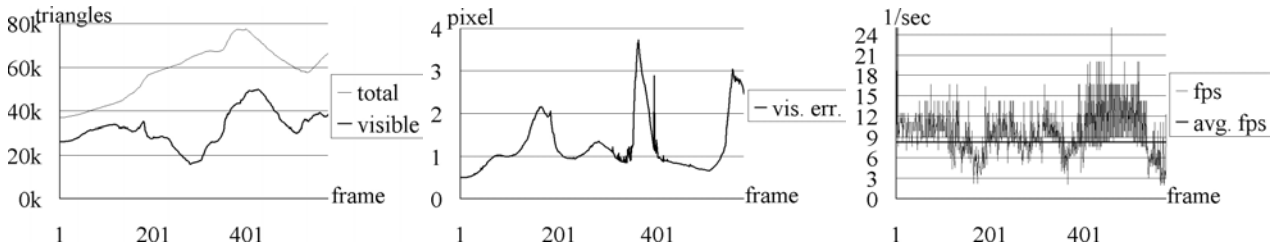
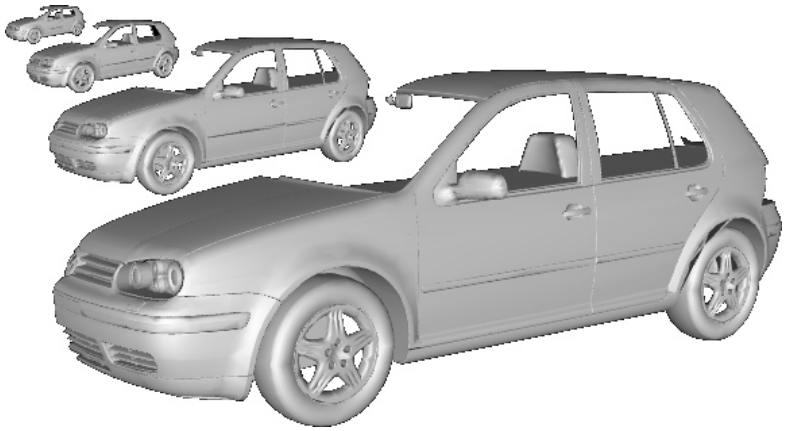



**Fig. 6.8: Rendering of the assembled car model.**

even at the slowest frame and the maximum visible error is acceptable, because it is removed within the next few frames (see Figure 6.8). Note that real-time rendering of models with about 3.6M triangles is challenging even for the most recently developed graphics cards.

Table 6.2 compares the complexity of models seen from varying distance using the Seam Graph and without it (i.e., using view-dependent tessellation and the highest level sewing points). Clearly, especially for coarse resolutions, the Seam Graph leads to significantly less complex triangular models and thus greatly increased frame rates.

The rendering algorithm allocates between 33.8 and 47.1 Bytes per vertex at maximum LOD for the tested datasets. This memory requirement consists of about 320 Bytes per boundary edge, about 2200 Bytes per trimmed NURBS patch (control points and knot vectors), 12 Bytes per visible triangle and 24 Bytes per visible vertex. Note, that the non-manifold multi-resolution data structure of Floriani et al. [148] requires 61.8 to 65.2 Bytes per vertex (including vertex normals).



| size | LOD tris | LOD pat. | no LOD tris |
|---|---|---|---|
| large | 37.2k | 3500 | 309k |
| med. | 18.2k | 3000 | 297k |
| small | 10.9k | 2500 | 292k |
| tiny | 6.8k | 2000 | 290k |

**Tab. 6.2: Number of displayed triangles and NURBS patches with and without LOD selection. Without LOD, always all 3914 patches are rendered.**

## 6.5 Conclusions

In this chapter the Seam Graph, a data structure for handling LODs of objects consisting of surfaces assembled in a non-manifold way, was introduced. It represents objects by their seams, i.e., the curves or lines that glue together the individual surfaces that it consists of. An application of the Seam Graph to efficient rendering of trimmed NURBS models was presented, but obviously many other surface types and application areas are possible. Especially important for other applications areas like finite element meshing is that the LODs encoded by the Seam Graph contain no cracks (i.e., watertight models remain watertight). This is in contrast to recently published approaches that efficiently hide cracks during rendering [18, 184].

The Seam Graph representation of a NURBS model is attractive since it retains the original surface representation, leading to two significant advantages over tessellated representations: First, it enables efficient generation of (arbitrarily) accurate models. Second, storage requirements are modest since on the one hand NURBS surfaces can be stored very efficiently, and on the other hand the Seam Graph LODs require less memory than conventional progressive mesh representations.

In the future, the Seam Graph and its applications could be extended in a number of ways. First, it would be very interesting to extend it to deformable surfaces, i.e., models with fixed connectivity but changing geometry. This would make the approach suitable for interactive visualization of editing operations. Currently, the trimming curves need to be fixed in space, which limits modeling capabilities significantly. Second, the Seam Graph could be applied to models containing different surface types (i.e., other parametric surface types or subdivision surfaces).

Third, better rendering results could be achieved by integrating more surface properties than geometry into the simplification (i.e., normals and texture coordinates). An

approach following this direction was presented by Guthe and Klein [188] who computed and compressed normal maps for patches. Additionally, faster rendering could be achieved by integrating the Seam Graph with the GPU rendering technique of Guthe et al. [184]: While the GPU technique is especially efficient for visualizing surfaces seen at close distance, its performance degrades when handling objects with tens or hundreds of thousands of visible patches (i.e., objects seen at a distance). Obviously, the Seam Graph is especially helpful in these cases, where a large number of patches can be removed due to coarse resolution display. Finally, the Seam Graph should be extended to handle out-of-core data, following schemes for out-of-core handling of progressive meshes, which were presented in the previous chapter.

# Chapter 7

# Billboard Clouds

In the previous chapter, a method for efficient visualization of CAD data was proposed. Typically, CAD models are used to derive geometrically accurate descriptions of object shape which serve as input to several validation processes and even manufacturing. Therefore, they rarely contain appearance attributes like color or even surface materials. Thus, predictive rendering from these models is usually not possible.

Nevertheless, geometrically accurate models are an essential ingredient for predictive rendering. Following the CAD modeling step, the objects are typically converted to triangle meshes, and materials are applied. Obviously, this increases the scene complexity even further, making the necessity to adjust the complexity of the model to its display size even more pressing. Chapter 5 briefly reviewed methods for simplification of meshes with surface materials. While excellent results can be achieved by preserving geometry (including orientation) and controlling texture deviation, true preservation of texture (i.e., spatially varying materials) remains difficult (cf. Figure 5.7). Among the existing methods, image-based object representations solve this problem most accurately by resampling surface materials for simplified models.

Recently, Décoret et al. [97] proposed Billboard Clouds (BCs) as an efficient image-based representation for rendering. The basic idea of the approach is to coarsely approximate the geometry of a rendered object by a set of quads and encoding fine-grained detail like surface roughness, surface color or the silhouette in normal and color textures (cf. Figure 7.1). Since current graphics boards are highly optimized to handle textures, such representations can be rendered very efficiently. Compared to other approaches, BCs achieve appearance preservation while at the same time requiring only very few geometric primitives to be rendered. Compared to other image-based object representations, the geometry of BCs is view-independent which removes the necessity for frequent updates in order to avoid image artifacts.

Like all image-based approaches, BCs tend to use much texture memory. Employing

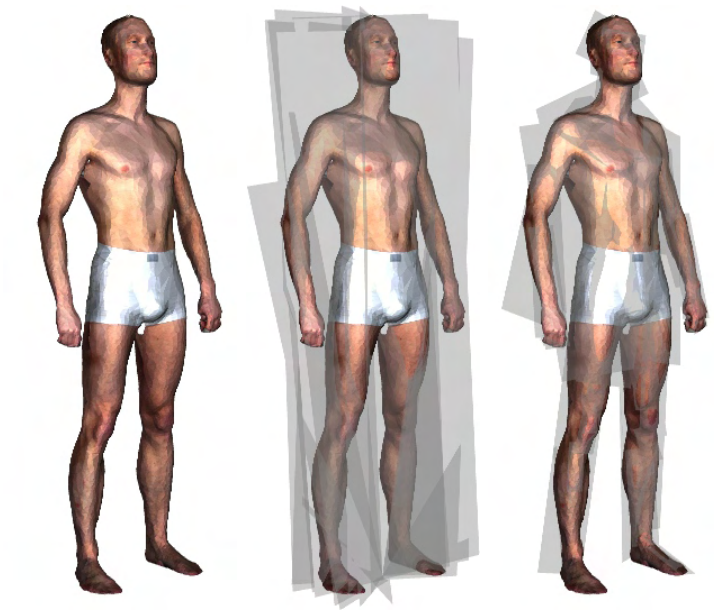


**Fig. 7.1: Billboard Clouds represent an object (left) by a set of textured quads. Different placement methods for the quads lead to different sets of BC quads (center: Hough space [97], right: face clustering).**

the BC generation method proposed by Décoret et al. this amount may well be a couple MBs for models projected to at most $100\times100$ pixels[1]. Therefore, BC construction methods should try to minimize these storage requirements.

In addition to the memory problem, two further deficiencies exist. The first will be called *view-independence* problem in the following: View- and light-dependent reflectance properties cannot be represented by a single texture and are therefore lost in the standard BC representation. In addition, other effects like view-dependent silhouettes and occlusions or light-dependent self-shadowing due to small surface detail are not preserved (cf. Figure 7.2). The second problem is the *normal sampling* problem: The normals of approximating quads may be completely different than the normals of the approximated faces which can lead to missing pixels in images (cf. Figure 7.4).

In this following, novel BC construction algorithms optimized for connected meshes and polygon soups are proposed that improve the abovementioned problems[2]. The normal sampling problem is solved by providing explicit control over normals represented by a plane. The view-independence problem is significantly improved by utilizing view- and light-dependent textures (i.e., bidirectional texture functions (BTFs)). Since BTFs require even more storage than textures, special care is taken to minimize the amount of

[1] Alternative representations like point-clouds require similar amounts of memory to achieve comparable visualization results. Tests showed that rendering the plant model in Figure 7.5 using QSplat [445] requires about 120k points for a displayed image of about $100\times100$ pixels resulting in about 500 kB of memory for storage of the points and their normals.

[2] The approaches were first published in [357].

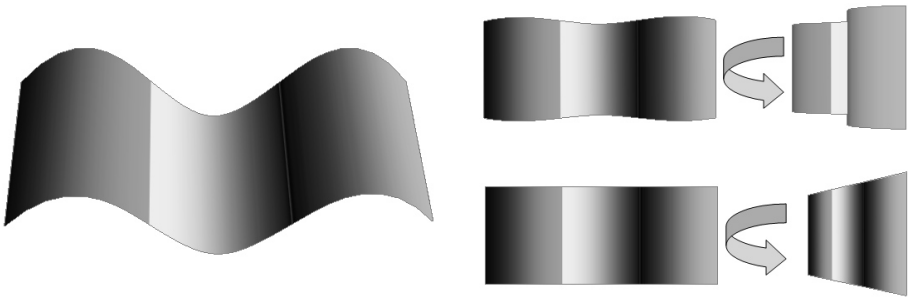


**Fig. 7.2: View-Independence Problem. Comparison of a curved surface (left and top) and an approximating textured quad (bottom). While the colors and the silhouette of the quad are correct for the frontal view (middle), they become incorrect for other view directions (right).**

required texture memory. In addition, the methods generate hierarchical BCs which can be employed for LOD rendering and easily extend to BC generation from point clouds.

The following text is structured as follows: First, a solution to the view-independence problem based on BTFs is proposed. Then, the BC construction technique of Décoret et al. [97] is described and analyzed, determining significant problems for practical use of this technique. To solve these problems, new methods for Billboard Cloud construction are proposed. Finally, the results achieved with these new methods are reviewed, conclusions about the novel techniques are drawn, and opportunities for future improvement are described.

## 7.1 BTF Textured Billboard Clouds

Billboard clouds (BCs) are sets of textured quads that approximate the geometry and surface detail of a model at a coarse LOD. Besides convincing renderings of the simplified model, this representation provides the possibility to efficiently cast approximate shadows since the silhouette of the object is well preserved. Unfortunately, since the silhouette is view-dependent (other than the BC geometry and its texture), such shadows will be incorrect for most light-directions. In addition, changing surface appearance due to reflectance properties of the surface material and simplified surface geometry cannot be preserved.

Such information can efficiently be stored as a view- and light-direction dependent texture, i.e., a BTF. Figure 7.3 shows the increased quality of BTF textured BCs compared to standard textured ones. Note that the BTF textured image shows better quality due to more precise reconstruction of the surface reflectance properties and the view-dependent silhouette.

Unlike the original intention of the BTF, which stores reflectance for a flat sample and therefore requires sampling of view- and light-directions on the hemisphere only,

**Fig. 7.3: Comparison of standard BC with texture and normal map (left) with BTF textured one (middle) and original model (right). The resolution of the texture and BTF are optimized for distant viewing (small images).**

the BTFs used for representations of BCs represent non-flat regions and therefore sampling of the complete sphere (for view-directions this can be neglected since rendered primitives will be invisible for view-directions outside the hemisphere). Hence, for this purpose the BTF is generalized to a 6D function $\mathrm{BTF}(\mathbf{x}, \mathbf{l}, \mathbf{v})$ of surface location $\mathbf{x}$, view direction $\mathbf{v}$ and light-direction $\mathbf{l}$ with $\mathbf{v} \in \Omega$ and $\mathbf{l} \in S^2$.

Construction of such BTFs can efficiently be done using rasterization hardware as in [97] for changing view- and light-directions in combination with a shadowing algorithms. Alternatively, a raytracer can be used. Resulting BTFs feature accurate sampling, which is especially important for scenes containing a large number of nearly arbitrarily oriented surfaces (e.g., the forest scene shown in Figure 7.9).

Since BTFs require much more memory than simple textures, one needs to choose a reasonable compression algorithm for rendering. Since surface geometry introduces significant and strongly varying effects like occlusions and shadowing, object BTFs are not well representable by per-pixel BRDF models as the one proposed in Chapter 3. Better results can be achieved using data-driven compression algorithms as the one proposed by Müller et al. [369]. In addition to efficiently compressing the BTF, it is essential to employ BC construction schemes that minimize the number of texels required for coding of surface detail. Such schemes reduce the amount of memory produced during BTF construction (which ranges up to several GBs) and reduce the compression error since additional texels tend to introduce additional variance into the BTF data.

## 7.2 Billboard Cloud Construction

Constructing optimal BCs is a very difficult since computationally expensive task. One needs to concurrently minimize:

1. the amount of memory for the textures,
2. the number of primitives that approximate the geometry, and
3. the loss of visual quality of the rendered model (which includes minimizing the view-independence and normal-sampling problems).

Since generation of optimal BCs is an NP hard problem [97], an optimal solution is unpractical. Therefore, greedy algorithms are employed.

### Hough Space

Décoret et al. [97] suggest to build BCs using the 3D equivalent of the Hough transform [227]: a plane is represented by the spherical coordinates $(\theta, \phi)$ of its normal and its distance $d$ to the origin. All faces of the original mesh are inserted into a regular grid which represents a spatial subdivision of 3D Hough space. For each cell $C$ of the grid *valid* and *missed* faces are determined. A face $F$ is considered valid with respect to $C$ if there exists a plane $P \in C$ such that the Euclidean distance between $P$ and the vertices of $F$ is smaller than the prescribed approximation error $\varepsilon_a$. $F$ is considered missed if there exists a plane $P \in C$ such that the distance between $P$ and the vertices of $F$ against the direction of the normal of $P$ is larger than $\varepsilon_a$ but smaller than $\varepsilon_a + \varepsilon_m$ where $\varepsilon_m$ can as well be chosen by the user.

Extraction of planes for the BC is done in a greedy fashion based on the accumulated, projected areas of valid and missed faces stored in the grid cells. After determination of each new plane, the contributions of faces within $\varepsilon_a$ distance of the new plane are removed from the grid cells. The process terminates when all faces are covered.

As a next step, for each extracted plane all points of the original mesh within $\varepsilon_a$ distance are projected onto it and the result is stored in a texture. To optimize texture use, textures are split into parts if unconnected, compact regions are detected.

The advantage of this approach is the applicability to arbitrary triangle soups and the small number of resulting textured quads (see Table 7.1). Unfortunately the approach does not explicitly handle the deviation of face normals from the normal of the textured quad which represents them. This can lead to bad results as in Figure 7.4, where many parallel planes were chosen to represent the geometry due to the lengthy shape of the

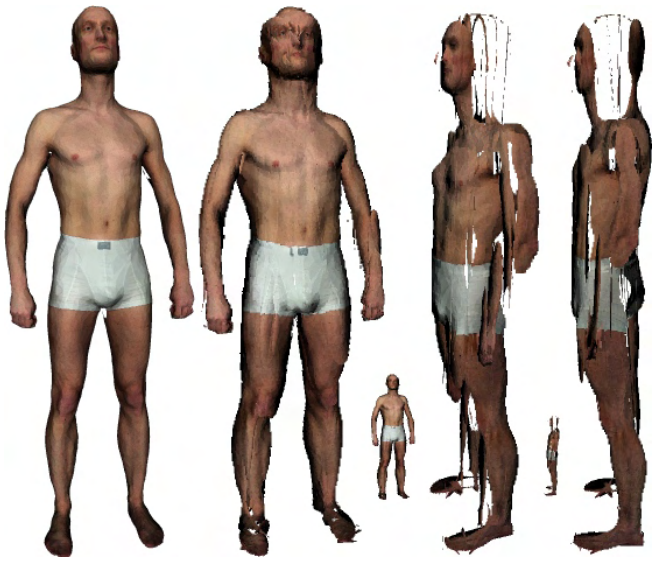


**Fig. 7.4: Normal sampling problem of the Hough space approach. While the frontal view of the BC (second from left) looks correct, pixels are missing when seen from the side even at the desired projection size (small images).**

| | plant | | | | Max Planck head | | | | man | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\varepsilon_{g_c}$ | 4 | 2 | 1 | 0.5 | 4 | 2 | 1 | 0.5 | 4 | 2 | 1 | 0.5 |
| Hough | 34 | 116 | 448 | 1.2k | 34 | 62 | 164 | 434 | 4 | 10 | 30 | 88 |
| | 29k | 334k | 2.9M | 22M | 30k | 175k | 1.6M | 14M | 0.9k | 6.1k | 67k | 600k |
| HFC | 200 | 566 | 1.5k | 1.9k | 60 | 93 | 193 | 478 | 99 | 102 | 121 | 200 |
| | 34k | 139k | 500k | 1.4M | 6.9k | 26k | 112k | 432k | 0.8k | 3.1k | 12k | 43k |
| Simpl. | 1.2k | 3.1k | 6.0k | 9.8k | 66 | 168 | 195 | 404 | 70 | 95 | 194 | 419 |
| | 2.5k | 17k | 67k | 313k | 4k | 17k | 70k | 280k | 0.6k | 3.3k | 24k | 150k |

**Tab. 7.1: Comparison of the BC construction algorithms. Per method, model and chosen geometric error $\varepsilon_{g_c}$ (specified as percentage of the longest side of the bounding box) the number of geometric primitives (top column) and the number of texture pixels (bottom column) is given.**

body. It is only due to the redundant sampling of surface points that few cases exist where the problem really becomes apparent.

Even worse, the method provides no control over the amount of required texture space since Hough space is insensitive to Euclidean distances and therefore it can easily occur that very distant geometries are represented by the same plane. The texture optimization approach proposed by Décoret et al. can resolve a very limited number of these cases only, leading to excessive texture memory requirements in many cases. Since texture space is already the limiting factor for application of BCs, other generation methods are required for reasonable simplification errors. In the following, two new techniques are represented which provide solutions to these problems.

## Mesh Simplification

The first method for memory-efficient construction of BCs is based on the standard way of reducing the complexity of models by successively removing vertices, edges, or faces of a given model: mesh simplification algorithms [329]. In contrast to the previous Hough space approach which extracts globally optimal planes, mesh simplification algorithms are usually based on local optimization.

In principle, standard mesh simplification algorithms allowing for topology simplification can be employed for determination of approximating geometry for BC generation. Yet, since an accurate evaluation of the geometric error between the original mesh and the simplified mesh is required, methods guaranteeing tight error bounds like the one of Borodin et al. [42] are preferred.

As Table 7.1 shows, applying these techniques to connected, smooth and preferably manifold meshes like the Max Planck head or the man yields models of rather low polygon count and minimal texture space requirements. In addition, the deviation of normals from the normal of the approximating triangle can simply be controlled [479].

For unconnected meshes like trees, mesh simplification algorithms performed much worse in the tests (although texture requirements remain very low): The number of triangles increases significantly compared to the Hough space approach and the results largely reduce in rendering quality due to the complex silhouettes of unconnected meshes which cannot be represented adequately. Enlarging the BC triangles to compensate for this problem unfortunately increases the texture requirements significantly (in the experiments the required texture space was doubled approximately).

## Hierarchical Face Clustering

The second new approach to BC generation is based on ideas from hierarchical face clustering [158]. The key idea of this method is to iteratively merge pairs of adjacent surface patches based on some energy function. Since face clustering is limited to connected meshes, a generalized definition of adjacency is used in the following.

### Spatial Proximity

One of the central points of hierarchical face clustering is the concentration on adjacent patches which is necessary for charting but hinders general BC construction. For the needs of BC construction algorithms, working on adjacent patches makes sense only insofar as this results in connected areas and therefore efficient texture use. Yet, since arbitrary meshes consist of many unconnected parts, the approach misses many perfectly

reasonable opportunities to merge geometry. Therefore, the notion of adjacency is generalized to spatial closeness which removes the limitation to connected input meshes. Spatial closeness can efficiently be computed even for large meshes using a spatial data structure (SDS) like a grid or an octree.

An important parameter for computation of spatially close parts is the definition of closeness. The most intuitive approach relates closeness to a maximum Euclidean distance. Following this definition, one has to compute all pairs of primitives that are no further apart than a predefined threshold. Unfortunately, to balance the amount of possible pairs during the merging process, this threshold needs to be increased over time. This increases the complexity of the algorithm.

Therefore, a primitive $p_1$ is defined to be close to another one $p_2$, if at most $n-1$ other primitives have a smaller Euclidean distance to $p_2$ than $p_1$. This methods leads to an easy yet efficient control of the number of possible merges and requires the user to define a single threshold only. Conducted experiments determined that an $n$ of around 50 leads to very good results, and that the accuracy improved at most slightly for higher values of $n$. Nevertheless, the exact parameter value varies from model to model since it depends on the model's geometric structure.

Another problem closely related to the run-time efficiency is the choice of an adequate number of spatial subdivisions introduced by the SDS. The tested implementation generates an initial subdivision based on the axis-aligned bounding volume of the model and the number of its primitives. This subdivision is adjusted at runtime (the spatial resolution in each dimension is halved as soon as the number of primitives reduces by a factor of eight) resulting in a good balancing of the number of subdivisions to the number of remaining primitives.

### Cost Function

Generalized hierarchical face clustering can be formulated as a minimization problem on the proximity graph. In the following, the proximity graph consists of nodes representing primitives of the object and weighted edges connecting spatially close primitives, where weights represent costs of merging two nodes. The graph is simplified by merging connected nodes until a predefined number of nodes remains or until any further merge operation requires costs above a predefined threshold. The sum of costs for simplification is to be minimized.

In contrast to standard face clustering approaches, which try to achieve well parameterizable, compact charts, a BC construction algorithm needs to minimize the amount of texture space. Therefore, the merge costs are combined of three different parts that measure geometric approximation error (i.e., maximum distance to a fitting plane), normal

deviation, and texture waste.

Given a pair of adjacent patches $(P_1, P_2)$, the geometric error $\varepsilon_g$ is simply half the length of the smallest side of the smallest oriented bounding box (OBB) containing $P_1$ and $P_2$. The OBB and the respective best approximating plane $P_a$ can efficiently be computed using principal component analysis. The normal deviation error $\varepsilon_n$ is defined as:

$$\varepsilon_n = \max\left\{\operatorname{acos}\left(n\left(F\right)\cdot n\left(P_a\right)\right)|F \in P_1 \cup P_2\right\}, \tag{7.1}$$

with $n$ denoting the normal of a face. Texture waste $\varepsilon_t$ is a more accurate version of the shape error from Garland et al. [158] optimized to the needs of a BC construction technique and is computed as:

$$\varepsilon_t = \frac{\sum_{F\in P_1\cup P_2}\operatorname{area}(F)}{\operatorname{area}(P_a)} \tag{7.2}$$

Since weighting these errors to compute the merge costs is a very difficult task (since appropriate weights are hard to determine), only one is utilized to define an ordering among the valid pairs while the others serve as hard rejection criteria defining validity. Since experience from LOD research shows that hierarchical LODs (HLODs) [128] lead to much better performance than continuous LODs, one only needs to be concerned about discrete LODs for hierarchical BCs. Therefore, setting a threshold for $\varepsilon_g$ for the most accurate HLOD level and doubling it for each coarser level is a reasonable choice that sorts possible merges into clusters assigned to the different HLOD levels. Since the major reason for including normal deviation in the evaluation is minimization of the normal sampling problem, $\varepsilon_n$ can be thresholded as well. A setting of $\varepsilon_{n_m} = 60°$, e.g., guarantees that no such problems can occur for spheres and cylinders.

Based on these definitions, the cost of a possible merge can be computed as

$$MergeCost = \begin{cases} \infty & \varepsilon_n > \varepsilon_{n_m} \\ \infty & \varepsilon_t > \varepsilon_{t_m} \\ \varepsilon_t + k\varepsilon_{t_m} & else \end{cases} \tag{7.3}$$

where $\varepsilon_{t_m}$ denotes the maximally allowed texture waste, and $k \in N$ is either zero if $\varepsilon_g \leq \varepsilon_{g_m}$ or otherwise determined by $2^{k-1}\varepsilon_{g_m} \leq \varepsilon_g \leq 2^k\varepsilon_{g_m}$.

Pairs resulting in too high normal deviation or texture waste are rejected. Pairs with valid normal deviation and texture waste are grouped into two categories: If the geometric error is below the threshold $\varepsilon_{g_m}$ for the finest HLOD level, their merge cost is equal to the texture waste. If the geometric error is above the threshold for the finest level, the

**Fig. 7.5: Plant model. Left and middle: original model (12k triangles). Right: BCs for $\varepsilon_g = 0.5\%$, 1% and 2% (cf. Table 7.1).**

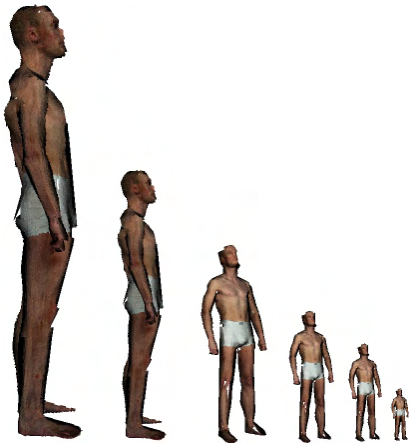


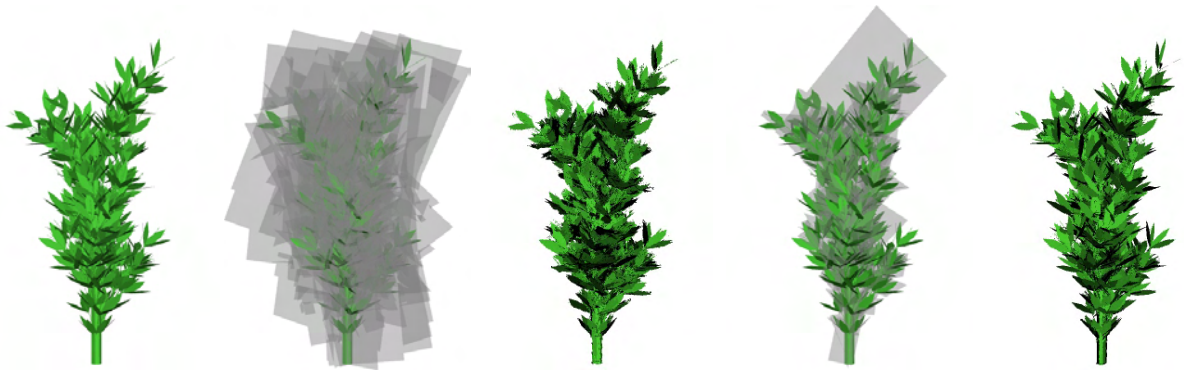
**Fig. 7.6: BC of man model from HFC ($\varepsilon_g = 1\%$, no BTF): the normal sampling problem is solved by setting $\varepsilon_{n_m} = 60°$. Incorrect silhouettes vanish as the desired projection size is approached.**

appropriate LOD level $k$ for which the geometric error is valid is determined and the cost is computed as $\varepsilon_t + k\varepsilon_{t_m}$. This assures that no such merge is executed before any possible merges from the previous LOD levels.

This new method has the big advantage that the required amount of texture space can be reduced significantly compared to the Hough space approach while preserving the high visual quality of resulting BCs (cf. Figure 7.7). In addition, generated BCs contain relatively few textured triangles and the inherent control over normal deviation eliminates most visible cases of the normal sampling problem.

## 7.3 Results

Multiple experiments were conducted with the BC construction methods described above. While standard hierarchical face clustering and mesh simplification of connected meshes turned out to be the fastest methods due to the restricted search-space (few seconds), the runtimes of the more general methods for unconnected models are slower (few minutes). As stated above already, the results generated by the methods vary greatly. In general, the Hough space approach requires the least number of primitives to represent the input model but fails to avoid the normal sampling problem and additionally requires the most texture space. For connected meshes like the man model in Figure 7.6, iterative mesh simplification and hierarchical face clustering generate results superior to the Hough space technique, with almost equal quality resulting from either method. For unconnected meshes like the plant in Figure 7.5, HFC turned out to



original Hough (448 tris, 2.9M texels) HFC (1.5k tris, 500k texels)

**Fig. 7.7: Comparison of visual quality of generated BCs (1% approximation error). Although the visual quality is very similar, the HFC BC requires 6 times less texture memory.**

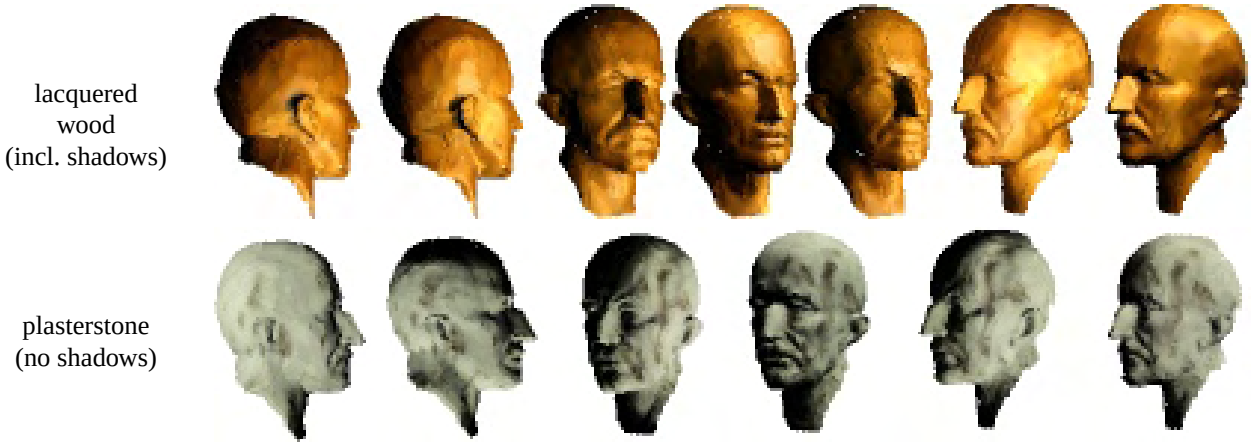



**Fig. 7.8: Views of BTF textured Max Planck head BC (123 textured quads) for varying view and light-directions. The appearance of the surface material varies drastically.**

be the most efficient BC generation strategy, since it leads to good quality results that require few geometric primitives and texture memory.

Computation of BTFs on the BCs was done using either rasterization or a simple ray-tracer. While rasterization provides run-time advantages (about 2 hours for the model in Figures 7.3 and 7.8 - bottom row), raytracing allows better quality due to pixel-correct shadows and interreflections (see top row of Figure 7.8 and Figure 7.9). Fortunately, both approaches can easily be parallelized.

The raw BTF requires about 30k memory per texel in the BC (about 1.8 GB for the model in Figures 7.3 and 7.8). Using the Hough space approach, about 18 GB would be required. The final amount of data per BTF textured BC was significantly reduced to about 17 MBs for the presented models by applying the compression method of Müller et al. [369], which enables efficient rendering using standard programmable graphics hardware [458].

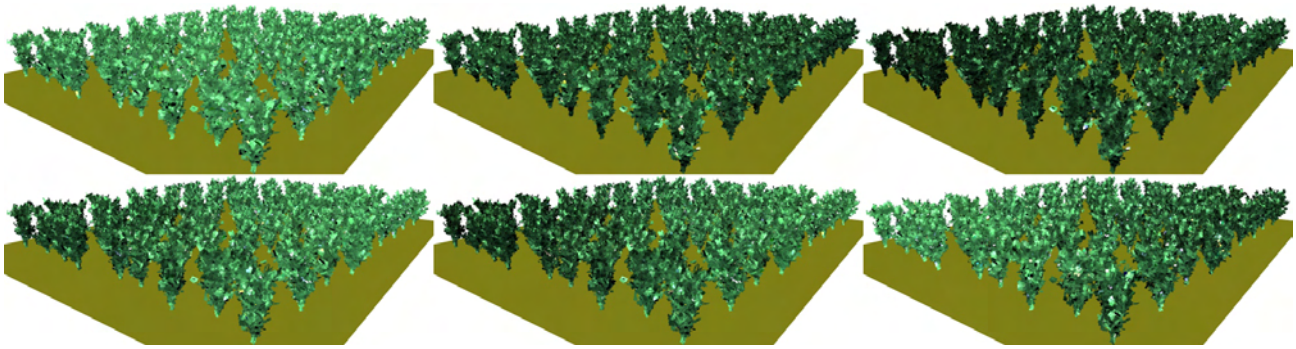


**Fig. 7.9: Relighting of distant wood scene. Occlusion and shadowing are correctly represented in the BTF and therefore the images of the distant trees due to the correct resampling used for BTF construction. Such results are not achievable with normal maps due to the large variance of normals per pixel even using improved techniques like mipmapping of normal maps [392].**

The HFC BC construction method can be applied to a large number of surface types. In the experiments, good results were achieved for connected and unconnected meshes, as well as for point clouds. Figure 7.10 compares the quality of images rendered from a point cloud of 150k points to the quality of a BC with 3.8k textured quads constructed with the HFC method. While the differences are clearly visible in closeups, differences become negligible for distant viewing. The images additionally show that the depth impression is well preserved when rotating the model.

## 7.4 Conclusions

The experimental results shown in this chapter indicate that Billboard Clouds are indeed a suitable representation for extremely simplified objects. Even highly complex materials like BTFs and intricate geometry are well preserved by a very small number of textured geometric primitives while only modest amounts of memory are required for storage. Reduction of geometric complexity works especially well since geometric features like holes or silhouettes are efficiently encoded as transparent textures.

The chapter introduced techniques to solve the problem that image-based object representations require frequent updates of either textures, geometry, or both. Solving the normal sampling problem removes view-dependent image artifacts due to large scale geometry. The solution to the view-independence problem eliminates problems due to small scale geometry. Unfortunately, the problem of extensive memory use was solved only partially, although significant improvements over previous techniques were achieved.

In the future, better solutions to this problem might be proposed. Especially important are approaches for memory-efficient encoding of multi-resolution object BTFs. As

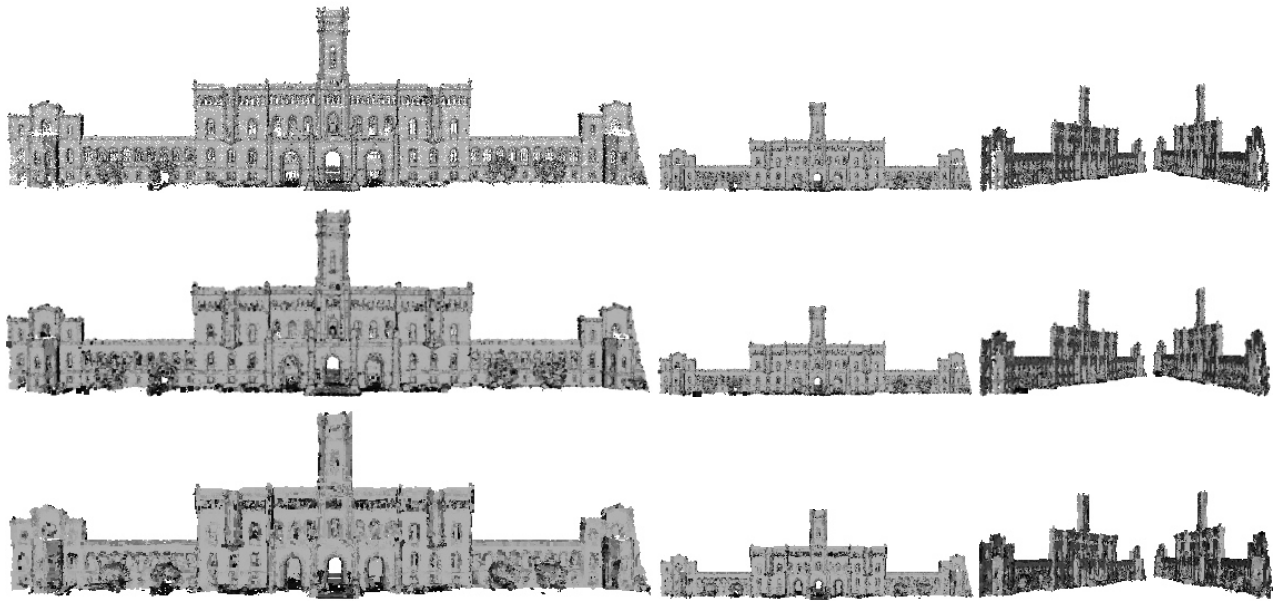


**Fig. 7.10: A castle point cloud with 150k points rendered with fixed point size (top), varying point size (middle), and as a BC with 3.8k quads (bottom).**

mentioned previously, the BTF contains effects from small-scale geometry. Obviously, the definition of a small scale depends on the display resolution and the projected size of the object. Therefore, BTFs at different scales contain very different geometric entities. Efficiently capturing and storing these (e.g., using view-dependent displacement techniques [556, 561]) seems to be an interesting research area for the future.

Another interesting point is the question whether Billboard Clouds are more efficient for representing extremely simplified objects than other representations. Recently, especially volumetric representations have been proposed [96, 163] for general objects. A related problem concerns the switching between respective LOD representations. So far, it is not clear at which point one should change from one representation to another, more efficient one. A simple approach was proposed by Behrendt et al. [25]: They cross-fade between meshes, Billboard Clouds, and volumetric representations. Obviously, fading introduces overheads which should optimally be avoided.

The technique presented in this chapter samples the visibility of points for a number of directions and compresses visibility using clustered PCA. While this reduces the data size and seems to lead to an acceptable solution, other approaches might give better results. Recently, Wu et al. [588] proposed an approach for view-dependent culling of texture pixels similar to the one presented in this section. Yet, their compression strategy is based on fitting spherical functions. Although their results do not seem to be very impressive (compression ratios of about 1:4) this indicates that research of different strategies might pay off.

Finally, in the context of predictive rendering it is absolutely necessary to evaluate the degree of realism of LOD representations. Mostly, evaluations are performed on abstract metrics only, omitting many significant influences. Other approaches try to determine the realism using psychophysical studies. Chapter 12 gives a brief overview of related techniques but none of them is capable of accurately predicting the suitability for neither photorealistic nor predictive rendering. Therefore, significant improvements have to be made in this area to make LOD models applicable in high-quality rendering applications. Yet, considering the current state of technology, techniques as presented in this chapter offer a very good compromise between achievable rendering speed and realism.

# Part IV

# Real-Time Physically-Based Rendering

# Chapter 8

# State of the Art

In the previous parts of this thesis methods for modeling of scenes such that efficient physically-based rendering becomes possible were presented. Part II concentrated on physically-based material representations, Part III presented methods for efficient handling of geometry, possibly in combination with materials, and only a detailed treatment of modeling of light source properties has not been given. Interested readers can find details related to realistic light sources in the course notes from Ian Ashdown [9], the recent book of Michael Goesele [164] (which originated from his PhD thesis) and in publications from industry producing devices for measurement of light source properties (e.g., goniometric radiometers).

With all the required inputs available, the scene is now ready to be rendered in an efficient, predictive way. This chapter presents on overview of existing methods related to predictive rendering, starting with general approaches that possibly do not achieve interactive performance, and concentrating on interactive methods in the following.

## 8.1 Physically Based Rendering

The goal of rendering is the transformation of a scene description into a 2D image corresponding to an observer's view of the scene. To achieve this goal the rendering algorithm has to compute the amount and spectral distribution of light reaching the observer's imaging organ or device. This, in turn, requires simulation of light transport from the light sources through the scene to the observer. If physically correct results are desired this simulation has to adhere strictly to the laws of physics. Since exact computation of physically correct results from general scenes seems impossible today and in the near future, physically based rendering methods typically consider only the physical effects most important to the perception of scenes. As an example, most ren-

dering approaches just consider geometric optics due to the high accuracy and largely increased efficiency of resulting algorithms.

Already in 1986 James Kajiya formulated a framework for computing light transport which became famous as the Rendering Equation [249]. While the original formulation was based on finite elements, the following reformulation became known more widely:

$$L_o(\mathbf{x}, \mathbf{v}) = L_e(\mathbf{x}, \mathbf{v}) + \int_{S^2} \rho(\mathbf{x}, \mathbf{v}, \mathbf{l}) L_i(\mathbf{x}, \mathbf{l}) \Big( \mathbf{n}(\mathbf{x}) \cdot \mathbf{l} \Big) d\mathbf{l}. \tag{8.1}$$

It states that the outgoing radiance $L_o$ of a (surface) point $\mathbf{x}$ into some direction $\mathbf{v}$ can be computed as the sum of radiance emitted by the surface point ($L_e$) and the reflected radiance. The reflected radiance can be computed from the incident illumination (the incident radiance $L_i$ from all directions $\mathbf{l}$ from the sphere $S^2$ of directions[1]), the material properties $\rho$ and the geometry orientation (specified by the normal vector $\mathbf{n}$). Unfortunately, solving the equation comprises recursions which leaves high-quality rendering algorithms with a large computational burden.

Existing approaches for solving the Rendering Equation can be grouped into two categories. On the one hand there are methods that compute the radiosity of finite element geometry patches. Motivated by related research on radiative heat transfer, radiosity methods were introduced to computer graphics in 1984 by the paper of Goral et al. [168]. Since these methods consider patches of geometry, their performance is relatively high for simple scenes, which made them very interesting in the 1980s and early 1990s.

On the other hand, there are methods directly implementing Equation 8.1 by considering samples of infinitely small extent. Such methods were first published in the paper of Whitted et al. [578] but many other publications dealing with raytracing approaches followed. Unlike radiosity methods, these approaches feature a greater flexibility since no patches of suitable geometric type and with homogeneous material have to be managed, which contributed significantly to their ongoing success.

This success is clearly related to the numerous extensions of the basic raytracing algorithm as proposed by Whitted et al. In 1984, Cook et al. [77] suggested distribution raytracing, which handles non-specular materials and area light sources by tracing several rays using Monte-Carlo sampling [146, 318]. The idea of using Monte-Carlo techniques to produce unbiased solutions was adopted by most following publications. In 1986, Kajiya introduced path tracing [249] which represented the first explicit, unbiased solution to the light transport equation. Two years later Ward et al. [566] proposed irradiance caching which allows for efficient handling of diffuse surfaces by storing and

[1] Most formulations of the Rendering Equation just consider the hemisphere of directions but this prohibits subsurface light transport effects.

reusing irradiance values. In 1994 another improvement of the rendering speed was achieved by proposing bidirectional path tracing (the method was proposed independently by Lafortune and Willems [289] and Veach and Guibas [533]). The approach enables efficient reuse of light- and camera paths and improves convergence in scenes with especially complicated lighting. In 1995 Pattanaik and Mudur proposed Particle Tracing [408] which is very similar to the Photon Tracing approach [238, 239] that Jensen presented in the same year. Like irradiance caching, these approaches aim at reusing computations by storing directed photons either in textures or scene-independent data structures. Today, raytracing-based techniques are the de-facto standard for computing high-quality images in the movie industry, for commercials, in offline product visualization, and many other business areas.

Obviously, numerous other improvements to the original schemes (radiosity and raytracing) were developed. For recent overviews of existing techniques laying special focus on Monte Carlo methods see the recent tutorials by Keller et al. [265] and Dutré et al. [119]. More comprehensive introductions are provided by the excellent books by Dutre et al. [118] and Pharr and Humphreys [412], where the latter focuses on implementing these approaches, e.g., accompanying a computer graphics lecture.

Despite the huge success of radiosity and especially raytracing methods for producing physically correct images, both approaches feature a major problem concerning their use in Virtual Reality: They are both computationally very demanding. Especially in combination with scenes containing many or big light sources and where materials with complex reflectance properties are used significant run-times are to be expected. Additionally, high-resolution display, which is typically required for realistic Virtual Reality, largely challenges the computation time for these algorithms. Therefore, these methods cannot be used for real-time physically-based rendering in Virtual Reality without explicit optimizations on standard computer configurations today. In the following section, such optimizations and other approaches for interactively computing approximations to physically based global illumination solutions are presented.

## 8.2 Real-Time Approaches

Unlike the methods for physically based rendering presented in the previous section, standard methods for real-time rendering as implemented in hardware by existing graphics boards handle local illumination models only. Typically simplistic lighting (either point or directional light sources, or lighting environments at infinite distance) and very restricted materials (e.g., combinations of Phong BRDFs, diffuse color textures and bump-maps) are supported, and integration of restricted global lighting phenomena like

hard shadows is possible. While the rendering speed achieved with such approaches is astonishing the rendering quality is typically far from being realistic (cf. Figure 1.1).

In the following, existing methods bridging the gap between hardly realistic real-time and slow, physically correct rendering methods are presented. Clearly many published approaches fall into this category. Therefore, only methods supporting more complex lighting situations or more complex materials than current standard real-time rendering methods are presented. Approaches for increasing rendering speed by optimizing geometry, e.g., using image-based impostors, will not be included since these were mentioned in Part III already.

Due to the large number of publications, the approaches are divided into five groups. The first group contains methods achieving interactivity by efficiently caching and reusing partial results. The second group includes methods based on advanced radiosity methods which allow for dynamic updates or non-diffuse materials, whereas the third comprises approaches for computing raytracing solutions interactively. The fourth group contains methods that mainly rely on real-time evaluation of precomputed (partial) global illumination solutions, which became very popular in the last years. Since these methods achieve good results for glossy-diffuse materials only, the fifth group covers methods for adding effects due to specular materials like reflections, transmissions or caustics.

### 8.2.1 Efficient Caching

Computing global illumination solutions requires significant time even if highly optimized methods are employed. E.g., the performance of raytracing methods is typically limited by the amount of pixels in the final image and the efforts spent per pixel. In many situations this seems inappropriate, e.g., when the resulting images show smoothly shaded continuous surfaces. In these cases similar results can be achieved by rendering the image at lower resolution and upsampling the image using some interpolation scheme.

#### Ray Caching

Methods following this approach were introduced by Teller et al. [516] and Bala et al. [15]. They try to identify regions of continuous outgoing radiance by estimating the shading error when interpolating radiance values from samples computed at the corners of geometric patches. If the error is below some user-specified threshold then interpolation from rays stored in a 4D data structure is performed. This way spatial coherence can be employed. Temporal coherence is achieved by reusing stored rays

for rendering consecutive frames. Unfortunately, the radiance interpolants used by both methods assume Whitted raytracing (i.e., indirect illumination for perfectly specular surfaces only) and it appears fairly difficult to generalize their approach to more general raytracing schemes. In addition, this approach does not guarantee any framerate.

A similar approach based on caching of rays was presented by Ward Larson [567]. Shot rays are stored in a 4D data structure. For arbitrary views of the scene the rendered image is reconstructed by selecting rays close to the current view point and view directions, and rendering the respective intersection points which are stored with the rays. Gaps in the images are filled based on Voronoi regions [194]. The image quality is improved progressively by requesting new rays based on least filled regions in the rendered image.

Ward and Simmons later improved this method in terms of rendering quality [564]. They use standard OpenGL rendering to display the geometry and extract depth discontinuities from the resulting depth buffer. Enforcing these discontinuities during the gap filling step significantly decreases blur in the resulting image. Additionally, they propose a gap filling algorithm based on Delaunay triangulation of the intersection points with known shading values which allows for smooth color blending between neighboring samples at the cost of maintaining valid triangulations.

This overhead was later reduced by Simmons and Séquin [484] who employ a fully adaptive mesh that requires minor updates per frame only. An extension of the scheme for requesting new rays by additionally considering color and depth differences of vertices of the mesh (which correspond to stored rays) improves the image quality faster than previously. Despite all these improvements the technique contains substantial management overhead for storing and retrieving rays (potentially from disk), managing Delaunay triangulations and estimating shading errors. Especially memory management problems appear to become more and more severe due to the gap in progression of CPU clock cycles and memory access time. Results can be produced at interactive framerates but presentation quality is severely limited in scenes with complex geometry, complex lighting, and glossy or specular materials. Finally, dynamic scenes are not supported.

### Sample Caching

Another group of approaches based on efficient caching are influenced by the idea of Frameless Rendering [31]: Instead of caching rays the shading values of individual pixels – which are computed from rendering processes executing global illumination solvers – are stored and reused by the display process.

The first method based on this idea is the *Render Cache* by Walter et al. [550]. One or several display threads constantly compute the coordinates and shading values of

3D points corresponding to image pixels and store them in the Render Cache. The display thread uses entries from this cache to generate images. Due to changing scene configurations, rendered images contain holes and invalid shading values (e.g., due to moving light sources or different camera positions). These are filled by requesting new samples. The request scheme considers the age of samples, color changes in the neighborhood and hints provided by the application (e.g., surfaces with glossy materials should have higher priority than diffuse surfaces). To achieve interactive speed and to fill remaining holes, images are computed at reduced resolution and gaps are filled by a $3\times3$ pixels filter sensitive to depth discontinuities. In a follow-up publication [549] Walter et al. improved the Render Cache by employing predictive sampling and cache-friendly data structures, deleting old points, using a combination of $3\times3$ and $7\times7$ filters for filling larger gaps while preserving the sharpness in densely sampled regions, and implementing several other optimizations. Unfortunately, the larger filter requires significantly more processing time and even the combination of filters cannot achieve hole-free images in all situations – especially if fast camera movement occurs.

A very similar approach was taken by Haber et al. [191] but their samples represent view-dependent illumination contributions only (view-independent effects are precomputed and rendered using rasterization hardware). Their scheme for aging samples bases on viewpoint changes. New samples are selected at various resolutions based on the perceptual importance of image regions. The approach features the same problems as the Rendering Cache: Holes or invalid shading are likely for objects with negligible view-independent illumination. Additionally, storage requirements for view-independent illumination contributions may be very high.

A slightly different approach was presented by Stamminger et al. [498]. Instead of caching points corresponding to pixels, they store global illumination information in *Corrective Textures* which are applied to the scene that gets rendered with standard OpenGL rendering techniques. Texture values and resolutions are updated based on a priority mechanism that pays attention to the materials' glossiness and the expected parallax error when texturing objects using plane-mapping. While the approach achieves hole-free images in all cases, the update rate of textures is quite limited. Additionally, texture management requires significant resources.

Similar to Simmons and Séquin, Tole et al. present a system that stores shaded samples as vertices of a refining or coarsening mesh. Their *Shading Cache* [518] updates samples based on a gloss-dependent aging scheme that estimates the expected shading errors by computing minimal and maximal color values of patches, patch areas, and region curvatures, and additionally tries to achieve an equal spatial sampling distribution. Compared to geometryless approaches like the Render Cache, storing samples as vertices of meshes leads to much better quality when moving objects and guarantees

hole-free images at the price of managing multi-resolution meshes.

A more recent improvement was published by Bala et al.: They propose computation of *Edge-and-Point Images* [16] which store discontinuity points and edges resulting from either illumination (shadows and material changes) or geometry (orientation or depth changes). These images are later used for restricting reconstruction filters to avoid sampling from samples on the opposite sides of discontinuity edges. While the resulting quality can clearly be improved in many cases, the approach cannot solve the problem that scenes with complex geometry, complex materials and complex lighting require a large amount of samples which cannot be produced interactively if global illumination algorithms are employed.

To reduce the number of samples, Dayal et al. [90] propose a spatio-temporal sampling and reconstruction strategy, which is supposed to replace existing sampling schedulers based on aging schemes. Choosing sample locations based on spatio-temporal color gradients and splatting sampled colors significantly increases image quality compared to aging schemes or sampling schemes neglecting temporal effects. Nevertheless, the approach is not efficient enough to handle arbitrary scenes interactively.

### Discussion

Development of methods based on ray caching started relatively early. The approaches implement a natural way to accelerating ray tracing by reusing shot rays. Unfortunately, a significant amount of accuracy is sacrificed if cached rays are reused and it is difficult to estimate the resulting shading errors efficiently. In addition, these techniques require high-dimensional caches with very many entries to be useful, which seems inappropriate given the currently existing gap between memory access time and the time to recompute intersections from scratch.

In contrast, methods based on sample caching feature many advantages: They support dynamic scenes, provide accurate guarantees on framerates and can be combined with arbitrary methods for computing illumination (typically physically-based rendering methods are used but even non-photorealistic rendering methods are possible). Unfortunately, despite all improvements published so far no guarantees with regard to rendering quality can be provided which is unacceptable for many applications of predictive Virtual Reality. Nevertheless methods for reusing (partial) shading results will most likely be part of future physically based rendering systems due to the tremendous opportunities for saving computation time by employing various sorts of coherence.

### 8.2.2 Radiosity-Based Methods

Classic radiosity methods as proposed by Goral et al. [168] compute global illumination solutions for scenes consisting of objects with diffuse materials only. While precomputation times for scenes with reasonable complexity are very high, walkthroughs are possible at real-time frame rates due to the view-independent radiosity solution. Recent publications additionally reported largely decreased precomputation times by exploiting the vast processing power of GPUs for solving the radiosity matrix equations [56] or even executing the entire radiosity algorithm on the GPU [80]. Unfortunately, the limitation to static scenes containing purely diffuse materials severely limits the applicability of this approach since such scenes represent largely simplified versions of real-world scenes.

#### Incremental, Hierarchical, and Clustered Radiosity

To extend the abilities of Radiosity, methods handling dynamic scenes – incremental radiosity and hierarchical (clustered) radiosity combined with special data structures for fast detection of necessary updates – and glossy materials – generalized form-factor computation – were proposed. For an overview of methods see the recent state of the art report by Damez et al. [86]. Nevertheless, satisfying results for scenes of industrially relevant complexity could not be achieved.

Recently, a novel method for computing global illumination solutions on the GPU inspired by clustered radiosity was published by Sunshine-Hill and Faloutsos [505]. They reformulate the rendering equation to operate on textures that represent the surfaces of parameterized objects in a scene. Lighting simulations can be performed by computing radiance exchange between texels, since they correspond to surface points. Naturally, computing radiance exchange between every pair of texels is prohibitively expensive. Therefore the authors select a number of 2D offsets $\delta_i$ and only compute radiance transfer between texel pairs $(t, t + \delta_i)$, where $t$ represents a texture coordinate. Missing interaction computations are accounted for by multiplying gathered radiance by the area of the Voronoi region of $\delta_i$. The authors achieve interactive results, e.g., for dynamically lighting a face and show the ability to support global light exchange effects in diffuse scenes including subsurface scattering effects. Yet, extensions for dynamic scenes and more complex reflectance properties seem very difficult.

#### Combinations with Raytracing

Very promising results were achieved by combining radiosity methods with other rendering approaches. Raytracing methods are fairly successful and efficient at simulating

global illumination in the presence of glossy or mirroring materials. Since radiosity methods lack this ability, many publications related to combining radiosity and raytracing appeared (e.g., [548, 482, 475, 476, 61, 483, 380]). The inherent difficulties include the clean separation of illumination effects captured by the radiosity and raytracing methods while reproducing interactions of diffuse and glossy surfaces in an as accurate as possible way. This separation is nicely implemented in the approach of Granier et al. [172]. It combines particle tracing and hierarchical clustered radiosity to include effects from objects with specular materials like caustics into dynamically changing scenes. Efficient updates of object positions and specular lighting effects is possible in combination with *shafts* [197]. Later the approach was extended in the work of Granier and Drettakis [171]. They obtained higher resolution specular effects without impacting the computation time of radiosity computations by employing caustic textures. The authors were able to render dynamic scenes of modest complexity at interactive frame rates. Unfortunately, application of these approaches to complex scenes appears very difficult due to the inherently quadratic (w.r.t. the number of geometric primitives) run-time performance of radiosity algorithms, even if clever optimizations are employed.

### Instant Radiosity Methods

Another very interesting and quite successful approach was presented by Keller [264]. His *Instant Radiosity* method approximates the indirect illumination in a scene by shooting a fixed number of photons from light sources. In the rendering stage the intersection points of photon-tracing paths and geometry are treated as virtual light sources. For each virtual light source, one image including shadows is rendered using rasterization graphics hardware and the resulting images are accumulated. The approach handles dynamic scenes naturally. Flickering artifacts due to quasi-random positions of virtual light sources in consecutive frames can be reduced by reusing photon paths over several frames. Unfortunately, the approach cannot handle specular surfaces and works in combination with glossy surfaces to some extent only since determination of photon-tracing paths and attenuation of the power of reflected photons bases on the average diffuse reflectivity of the scene. Very recently, a bidirectional version of Instant Radiosity was proposed by Segovia et al. [465]. Virtual light sources are additionally created by tracing camera paths of length two to obtain locations that will have an impact for visible surfaces. Only a subset of the created light sources is used for rendering, keeping those locations potentially most relevant for the current settings. Although less efficient than the original Instant Radiosity method, a GPU implementation of the bidirectional approach achieves interactive to real-time frame rates.

An approach similar to Instant Radiosity was published by Udeshi and Hansen [527].

Their approach renders accurate direct lighting including shadows using standard graphics hardware and adds approximate one-bounce indirect lighting. The algorithm accounts for indirect lighting from diffuse surfaces by computing approximate form factors between rendered surfaces and a number of polygons in the scene which are automatically selected as virtual light sources based on the amount of light they reflect. Indirect illumination from specular surfaces is computed by standard raytracing algorithms on the CPU. Due to the necessary computations required to select the brightest polygons and to perform raytracing, multiple CPUs are required to achieve interactive performance. The rendering from many light sources is accelerated using multiple rendering pipes and omitting shadow computations for virtual lights.

Recently, an algorithm motivated by Instant Radiosity but running on the GPU was presented by Dachsbacher and Stamminger [83]. They extend standard shadow maps in such a way that lit points in the shadow map act as point light sources for indirect illumination. During rendering, indirect illumination of fragments is approximated by sampling light contributions from pixels in the shadow map that are close to the projected position of the rendered fragment in the shadow map. Although efficient determination of visibility of lit points in the shadow map is difficult and therefore omitted, the approach yields a reasonable approximation of first bounce indirect light. Self-shadowing can be approximated with ambient occlusion techniques [292]. For moderately complex scenes and resolutions the method achieves interactive to real-time frame rates on a single GPU. Unfortunately, materials in the scene need to be approximately diffuse. The approach was recently improved [84] by switching from a gathering to a shooting approach implemented in a deferred shading pass, which includes efficient importance sampling of light sources causing indirect lighting. This also enables more efficient handling of glossy surfaces, making effects like caustics possible.

Another method influenced by Instant Radiosity is *Selective Photon Tracing* [111] by Dmitriev et al. Unlike the Instant Radiosity approach, which invalidates photon paths based on age, Selective Photon Tracing tries to identify invalid paths based on objects that moved in the scene. The method starts by shooting photons according to the quasi-random Halton sequence and storing these in a photon map. Whenever objects are moved *pilot photons* detect groups of invalid photon paths which are finally corrected by shooting photons with negative energy. The periodicity property of the Halton sequence enables detection of correct parameters for shooting corrective photons without storing photon paths in special data structures. The approach achieves interactive frame rates for moderately to fairly complex scenes on a single PC. In a following publication by Jiménez et al. [243] the approach was extended to handle participating media at comparable frame rates.

### Radiance Caching

The last group of interactive radiosity algorithms exploits the (local) smoothness of incoming radiosity encountered in many scenes, which was initially used for irradiance caching [566]. A first approach was introduced by Greger et al. [176]: They compute a global illumination solution and store incoming radiance values for vertices of a grid subdividing the scene, and a number of sample directions. This enables rendering of moving objects by interpolating radiance values for every point in space from the closest grid vertices and sample directions. Illumination effects of the moving object are neglected which is reasonable for objects with rather small extent.

The approach was improved in terms of efficiency by Nijasure et al. [390]. Instead of storing sampled directions they store incoming radiance as coefficients of spherical harmonics basis functions, which enables efficient shading of glossy-diffuse BRDFs. Direct illumination is rendered using standard OpenGL in combination with shadow mapping. The authors present an efficient scheme for dynamic recomputation of incoming radiance at the vertices of the grid by rendering the surrounding environments into cube maps and projecting these into spherical harmonics. A slightly more sophisticated technique was employed in a recent publication of Gautron et al. [160]: Instead of caching radiance at grid vertices, it is stored on surfaces as in the original irradiance caching method. Additionally, higher-quality samples are computed by Monte-Carlo raytracing, and interpolation is improved by introducing translational radiance gradients. Unfortunately, the respective radiance update scheme is less efficient than the one of Nijasure et al., practically limiting the method to walkthroughs of static scenes.

### Discussion

Radiosity methods were originally developed to allow interactive walkthroughs of static, diffuse environments. Several improvements of the original method gradually improved the capabilities, handling dynamic scenes, exchangeable illumination, and even exchangeable materials. Unfortunately, two major problems still remain: Handling of glossy-specular materials is hardly possible, and scenes with industrially relevant size are very difficult to deal with. Even methods based on Instant Radiosity cannot remove these limitations, although these methods should probably be considered the most advanced and successful.

Due to the lasting trend towards sampling based approaches for computing global illumination, it seems likely that the importance of radiosity methods will be reduced even further in the future. Nevertheless, the continuous development of novel methods shows that it still represents an important technique.

### 8.2.3 Interactive Raytracing

As stated above, raytracing methods are the de-facto standard for computing high-quality images and animations. Unfortunately, standard methods require very long processing time due to the large amount of rays that need to be traced when computing unbiased high-quality images with low variance.

Within the last years several approaches for bridging this gap appeared. Almost all of them tried to exploit the inherent parallelism of raytracing. While the earliest methods employed multi-CPU systems, other methods based on custom raytracing hardware exist as well. Additionally, many researchers successfully ported raytracing onto existing programmable, highly parallel GPUs. In the following, brief descriptions of existing methods from these three categories are provided. Further details can be found in the state of the art reports of Wald et al. [546, 545] and the PhD thesis of Ingo Wald [537].

#### Raytracing with CPU Clusters

The idea of exploiting the vast parallelism of raytracing algorithms for achieving interactive rendering was first used in the publication of Muuss et al. [373]. They presented a system running on an expensive shared-memory supercomputer that achieved interactive ray-tracing of very complex constructive solid geometry environments by distributing raytracing over several CPUs. Since their target application was simulation of radar-like effects for military operations, they did not concentrate on accurate indirect illumination effects.

Three years later, Parker et al. [404] suggested a raytracing system for interactive isosurface rendering from uniformly sampled volume data running also on a multiprocessor shared-memory supercomputer. Their system already contained some performance optimizations. First, they introduced a two-level grid as spatial datastructure, which allowed for efficient empty space skipping. Second and probably more importantly, they recognized the extreme importance of optimizing cache performance, which is a limiting factor of existing raytracing methods. To account for this problem, they suggested a rearrangement of the data based on volumetric tiles. Load-balancing was achieved by assigning tiles of the output image to available CPUs.

In a following publication [405] Parker et al. extended their system to a full-featured recursive raytracer handling triangular models and spline-based surfaces (e.g., NURBS). Their approach featured accurate shadows and reflections, and enabled texturing of surfaces. In addition, they introduced a more sophisticated load-balancing scheme that achieved better CPU utilization.

Another two years later, Wald et al. published a very insightful paper [539] on interactive raytracing. They recognized that tracing packets of four rays instead of a single ray

is more efficient on existing CPUs with SIMD instructions if the rays are spatially close. While this idea clearly reduces the number of executed instructions, the main improvement was found to be the improved cache coherency since spatially close rays are very likely to access similar data. Besides this observation, Wald et al. optimized their raytracing algorithm by changing recursive methods into iterative ones and limiting their code to handling triangular surfaces only. They showed that these improvements can increase the performance by an order of magnitude and finally suggested an implementation on a cluster of PCs. Compared to previous implementations on shared-memory supercomputers, this solution introduces much fewer costs for necessary hardware, is more easily extensible, but suffers from much higher latencies when communicating between participating CPUs. To reduce the latency problem, Wald et al. [547, 545] proposed an efficient distributed data loading scheme (which nevertheless limits the size of the handled model to the amount of main memory in each participating PC) and a tile-based load-balancing scheme. A recent publication by Reshetov et al. [437] achieved a further significant improvement in rendering performance by efficiently determining suitable entry points into the spatial subdivision hierarchy. They propose to intersect view-frusta, representing spatially close groups of rays, with filled nodes in the hierarchy, and splitting frusta if the contained rays intersect different nodes.

While raytracing systems implementing these ideas achieve interactive performance for large static scenes, they are still missing several features compared to existing APIs based on rasterization. To support raytracing of dynamic scenes at relatively small overhead Wald et al. [540] extended ideas from Lext et al. [308] and proposed a hierarchical spatial subdivision datastructure. Recently, novel schemes for raytracing dynamic scenes based on bounding-volume hierarchies [541], generalized kd-trees [180, 536], or a novel grid traversal algorithm [543] were introduced. Nevertheless, fully-dynamic scenes are still not possible due to the large overhead of updating or rebuilding global data structures. Support for highly complex models was first introduced by DeMarle et al. [101] who proposed a distributed memory management scheme for uniformly sampled volumetric data-sets. Wald et al. [542] later introduced an out-of-core data management scheme for triangular data. They propose to hide loading latencies by visualizing impostor geometry whenever the actual data is not available in memory.

Other publications focused on supporting more complex types of geometry for interactive ray-tracing. Very efficient algorithms for (trimmed) Bézier surfaces [29, 150, 161, 30], NURBS surfaces [121] and subdivision surfaces [29] were published recently. Additionally, researchers start to incorporate LOD techniques into raytracing algorithms since they reduce the memory footprint of large scenes and significantly reduce aliasing problems [602].

Another major limiting factor of interactive raytracing applications as presented so

far is lack of global illumination effects, since they implement raytracing in the way proposed by Whitted et al. [578]. To overcome this limitation, Wald et al. [544] combined Keller's Instant Radiosity method [264] with an interactive raytracing system. This not only eliminated the need for multi-pass rasterization rendering but additionally introduced all lighting effects achievable with raytracing methods (i.e., reflections, refractions, and caustics). Since it became necessary to trace a much larger number of rays compared to the previous raytracing implementation, the authors proposed a scheme for increasing performance. Instead of managing and raytracing a single large set of virtual light sources, they suggested management of nine smaller sets of virtual light sources. They divided the rendered image into tiles of $3 \times 3$ pixels and considered a different set of virtual light sources for each of the pixels in such a tile. To remove the artifacts resulting from different light source sets for neighboring pixels, they introduced an image-space filter respecting discontinuities in geometry and shading similar to the one proposed by Bala et al. [16]. Later Benthin et al. [28] optimized the interactive raytracing kernel by tracing packets of rays with corresponding virtual light source sets from neighboring tiles. Additionally, they introduced a streaming method for accelerating shading computations. Optimized support for caustics was later added by Günther et al. [181]. They increased photon mapping performance by a fast density estimation algorithm, an approach to reduce the amount of data to be transferred between the cluster PCs, and avoiding shooting photons that do not contribute to caustics by adapting the pilot photon strategy [111].

An alternative approach to implementing raytracing on a cluster of CPUs in a brute-force manner is the use of progressive ray tracing [237, 107, 403] techniques. These try to minimize the number of shot rays by detecting areas with smooth shading and adaptively shooting rays in areas with discontinuities. Thus they resemble approaches that employ Radiance Interpolants to reuse cached rays [516, 15].

This scheme was employed in the *Vertex Tracing* method of Ullmann et al. [530]. Their approach starts by shooting rays from vertices of visible triangle edges (which already saves computation of first hit intersections at the cost of determining edge visibility) and computing illumination contributions due to specular materials. These indirect illumination contributions are interpolated over the triangle by rendering meshes with Gouraud shading and adding direct illumination. The solution can be refined progressively by subdividing edges with highest discontinuities in indirect illumination values. Later an extension [529] distributing ray tracing over several processors for an almost linear speedup due to small communication overhead and suitable load balancing was presented. While the approach achieves good results for some application scenarios and is employed in industry already, it is much less general than other methods for interactive raytracing. Additionally, it features the problem of determining edge visibility

which is implemented by rendering color-coded vertices, examining the frame buffer, and selecting all edges adjacent to a visible vertex. This does not only introduce quantization problems when rendering vertex IDs and requires costly readback of the frame buffer but additionally misses some visible triangles since triangles might still be visible if the adjacent vertices are hidden. Therefore, the approach appears to be inferior to other methods utilizing clusters of CPUs.

### Raytracing with Customized Hardware

The basic raytracing algorithm as presented by Whitted et al. [578] is very simple to implement in software. Especially core functions like ray-triangle intersection tests can be implemented by a very small number of instructions. Therefore, it is hardly surprising that several researchers decided to implement the intersection test or even a full raytracing algorithm in hardware.

First such approaches originated from volume visualization. While the VIRIM system [182] is based on digital signal processors (DSPs) and was built as a separate machine, the VIZARD [274] and VIZARD II [354] systems are mainly based on field-programmable gate arrays (FPGAs) and were built as a PCI card solution. As a result of concurrent work the VolumePro hardware [411] – based on application specific integrated circuits (ASICs) mounted on a PCI card – was presented. The latter cards allowed for interactive to real-time volume rendering of data sets of sizes up to $256^3$ voxels including gradient and shadow computation.

The first publication targeting raytracing of surface models was by Humphreys and Ananian [228]. They proposed a hardware architecture based on low-cost DSPs for efficiently computing intersections between rays and triangles in a scene. Yet, it seems that a card based on this architecture has never been built or used.

At about the same time Advanced Rendering Technology started the development of a first commercial raytracing accelerator chip: the AR250. The raytracing processor was able to compute intersection and geometry computations. The successor, the AR350 [198], is now available as part of a commercial hardware raytracer and can, e.g., be used for computing images using path tracing [57]. Yet, the AR350 processor is currently not targeted for interactive rendering.

Somewhat later Purcell [426] developed SHARP, an architecture for interactive raytracing based on *smart memories* [337], i.e., multi-processor shared-memory systems manufactured on a single chip. Individual processors act as ray generators, ray traversers, intersection computers, and shading units, allowing for a highly flexible system. Unfortunately, despite its indisputable capabilities the system never became available commercially.

With FPGAs becoming more widespread for testing hardware implementations and replacing core software functions, Fender and Rose [139] presented the design of a simple yet functional implementation of ray tracing on FPGAs. They employed clever fixed-point arithmetic to avoid complex floating point units and support spatial subdivision by a bounding box hierarchy with depth three. The simulated prototype achieves interactive performance for moderately complex scenes and moderate resolutions on an FPGA clocked at 100 MHz. The authors additionally conclude that highly interactive results should be possible for the same scenes and resolutions when using six units in parallel running at doubled clock frequencies.

Almost in parallel, Schmittler et al. [455] presented a simulation of a custom-built raytracing chip. Following the arguments from Wald et al. [539], they identify memory bandwidth as the most limiting factor of ray tracing applications and therefore advocate tracing packets of 64 adjacent rays to improve coherent data access. The suggested architecture hides latencies when loading data from on-board memory using hardware-supported multi-threading with up to 16 concurrent threads. Simulated real-time performance is achieved by deep pipelines resembling the fixed-function pipeline of existing GPUs and relatively large on-chip intersection caches. The design was later extended by a virtual memory management system [454] in combination with compact on-chip caches which reduces the necessity for large and costly on-board memories. A number of simulations of the modified design show only small overheads due to more flexible memory management when using multi-threading. Schmittler et al. [456] later published an implementation of the simulated architecture on a single FPGA chip that additionally supports dynamic scenes in the way proposed by Wald et al. [540]. Even for this single, prototypical chip they achieved interactive performance for scenes of almost arbitrary complexity at standard resolutions. Yet, their results contain direct illumination effects only.

In a following publication Woop et al. [587] presented a complete redesign of the previous architecture also implemented on FPGAs. Their ray processing unit consists of three main parts: fixed function ray traversal units supporting kd-trees, a mailboxing unit to reduce the number of intersection tests, and fully programmable shading units. The design is largely influenced by existing programmable GPUs but allows for more complex flow control such that recursive ray tracing becomes possible. It supports procedural geometry, procedural lighting, and programmable materials. Compared to the approach of Schmittler et al., the rendering performance is reduced due to the greater flexibility of shading computations. Nevertheless, the authors assume that commercial products could place multiple ray-processing units on a single chip or board, which should lead to a significant improvement in rendering performance. Yet, problems due to high memory bandwidth requirements would have to be solved.

In summary, accelerating ray tracing by dedicated hardware is currently an interesting and active area of research. Impressive results were achieved in the last years already but significant improvements will be necessary in the future to enable interactive, high-resolution rendering including full global illumination effects. Especially the problem of almost random memory access patterns when computing indirect illumination effects, which results in very bad cache performance, will require special attention.

### Raytracing on the GPU

Unlike researchers proposing new hardware architectures for efficient raytracing, several publications concentrated on porting raytracing onto existing specialized hardware, namely GPUs. Existing GPUs are highly parallel SIMD processors with up to 320 independent processing units with simple instruction sets[2]. Especially important for raytracing, their raw floating-point processing power clearly outrivals the one of CPUs and is expected to grow much faster in the future than the one of CPUs. Therefore, despite their limited instruction set they offer a great potential for increasing raytracing performance.

The first publication advocating use of GPUs for efficient raytracing was by Purcell et al. [428]. Due to missing hardware features of GPUs at the time of publication, they performed a simulation of raytracing on the GPU only (note: all the missing features became available on GPUs later). They divided raytracing into four stages: setup of traced rays, ray traversal through a regular grid until cells containing geometry are found, ray-triangle intersection and determination of closest hit points, and finally computation of outgoing radiance (i.e., shading). All stages are implemented as fragment programs that get triggered by rasterizing screen-sized quads. Inputs to and outputs of the stages are stored in textures, making the approach a stream-processor.

Later, implementations of this approach were published [253, 66] which produced results comparable to the performance of standard raytracing on the CPU for tiny to complex scenes. Additionally, improvements to the method were proposed. Wood et al. [585] suggested an extension to Purcell et al.'s approach that enabled handling of planes and quadric objects by defining separate fragment shaders for these surface types. Following their approach, acceleration data structures are defined per surface type, i.e., they do not use any spatial data structure for planes and quadric objects. A very efficient scheme for raytracing animated meshes with fixed topology represented as geometry images [178] was presented by Carr et al. Another approach using the regular structure of image-based representations was proposed by Krüger and Westermann [283]. They first convert the scene as seen from the camera into a layered depth image [466] and

[2]Number from ATI RADEON$^{TM}$HD 3850 data sheet

then efficiently determine intersections by rasterizing lines. Unfortunately, they cannot capture influences of elements outside the view frustum.

Two other publications concentrated on replacing the uniform grid used in Purcell et al.'s approach. While grids give rise to very efficient traversal [5], their performance for scenes with non-uniform distributions of geometry was shown to be inferior to hierarchical spatial data structures [207]. Therefore, Foley and Sugerman [149] investigated efficient methods for kd-tree traversal on GPUs (note: current GPUs do not support recursion). They came up with two approaches, one requiring additional storage of links to father nodes in the kd-tree, the second one based on restarting from the root node whenever a leaf node was found to contain no intersection. Not surprisingly, performance improved compared to a regular grid for most types of scenes. In the following, Thrane and Simonsen [517] compared the performance of Foley and Sugerman's kd-trees to hierarchies of axis-aligned bounding volumes. Very surprisingly they discovered that bounding volume hierarchies lead to much better performance than kd-trees which contradicts Havran's [207] evaluations for raytracing on the CPU. Although Thrane and Simonsen try to conduct a fair comparison, these results need to be validated by other publications in the future.

Another approach towards fast raytracing on the GPU by Carr et al. [55] only ports computation of ray-triangle intersections onto the GPU, while remaining operations are handled on the CPU. Their implementation shows a speedup compared to pure CPU based raytracing but unfortunately requires costly readback of texture data. A combination of Purcell et al.'s and Carr et al.'s approach was successfully employed by Weiskopf et al. [575] to compute non-linear raytracing. The traversal stage from Purcell et al.'s approach was modified such that rays travel along curves defined by ordinary differential equations using an adaptive step size control. Ray-object intersections are computed using Carr et al.'s approach. Unfortunately, realistic results cannot be computed at interactive frame rates even for small scenes.

Other publications aim at porting photon mapping onto the GPU. A first approach was presented by Purcell et al. [429] which represents a two-step version of their GPU raytracing approach [428]. The first step consists of the four stages photon generation, photon tracing, intersection determination, and photon storage in a regular 3D grid. In a second step view-rays are generated, traced through the scene, closest intersection points are determined, and hit-points are shaded. Shading includes a final gathering step which selects close photons in the photon map. Due to efficiency reasons, only very small regions are used for gathering. Since the 3D grid contains no information concerning orientation of intersection surfaces, errors may occur when gathering photons from neighboring surfaces with significantly different orientation.

Czuczor et al. [82] aimed at improving the non-interactive performance of Purcell et

al.'s approach. They proposed to reduce final gathering to filtering of a photon map texture atlas, where tiles in the atlas correspond to surfaces with similar orientation. Therefore, a presegmentation of the scene is necessary. The authors implement filtering either by fragment shaders requiring very large numbers of texture lookups per fragment, or by a very efficient implementation that generates very crude results for non-diffuse surfaces. Unfortunately, the presented approach does not work for smooth objects like spheres or cylinders for which no parameterization can be found such that points adjacent in object space remain adjacent in texture space.

A different approach towards global illumination was presented by Hachisuka [193]. Indirect illumination is computed following a proposal of Szirmay-Kalos and Purgathofer [510]: Instead of gathering incoming radiance using the hemicube-method [72] which takes a large number of samples for every point, sample directions from a precomputed set are chosen. Since this set contains much fewer entries, the number of samples to compute is reduced efficiently. Ray tracing for the fixed directions from the set can efficiently be computed by rendering all depth layers of the scene for fixed view directions and parallel projection using graphics hardware. The approach efficiently evaluates this information and supports multi-bounce indirect illumination by iterating this scheme. Unfortunately, the approach does not achieve interactive performance.

While Hachisuka aimed at high-quality global illumination solutions at non-interactive frame rates, Larsen and Christensen [293] sacrifice correctness for achieving real-time results. They use photon mapping to approximate caustics and indirect illumination of diffuse surfaces while direct illumination is rendered using standard OpenGL rendering including shadow mapping and approximate specular reflections from dynamic environment maps. Photon mapping for indirect illumination is computed by tracing small groups of photons on the CPU similar to the scheme of Dmitriev et al. [111]. Efficient final gathering on the GPU is performed by subdividing the scene into patches and computing a low-resolution irradiance map by evaluating incoming radiance for a sparse grid of positions on that patch using a simplified hemi-cube approach [482]. Efficient caustic rendering is achieved similar to the approach of Stürzlinger and Bastos [502] by tracing a small number of photons on the CPU and splatting these into surface textures.

While implementations of raytracing on the GPU create impressive results that challenge highly optimized implementations on CPUs [427] already, this comparison could turn out even more positive in the future due to the faster growth of GPU computation power compared to CPUs. Especially interesting for many application areas is the possibility of tightly integrating GPU raytracing algorithms and GPU rasterization techniques which are very wide-spread today. Some of the above techniques simply fit into a rasterization framework by employing specialized shaders.

Nevertheless, existing approaches also have to face several problems. Most importantly the size of scenes handled by most methods is limited by the rather small memories graphics boards are equipped with. Developing virtual memory management methods on GPUs was investigated already [299] but it remains questionable if resulting latencies can be hidden efficiently. Other problems result from the limited instruction sets of GPU stream processing units (especially random-access write operations are not possible). Some of these should be resolved in the near future but others will remain since memory management becomes much more complex if several units can write to memory concurrently.

#### Discussion

During the last ten years tremendous improvements towards real-time raytracing including global illumination effects have been made. Solutions were implemented on clusters of CPUs, specialized hardware, and GPUs, all of them having specific advantages and drawbacks. The results show that interactive ray-tracing is possible today. Yet, the presented methods still cannot reproduce all global illumination effects fast enough, especially if high-resolution output images (e.g., for Caves or Powerwalls) are required. Especially handling of glossy-diffuse materials and area light sources, which increases the number of traced rays significantly, is not possible today. In addition, volumetric effects from participating media like subsurface scattering or fog need to be made possible.

Despite the remaining limitations, raytracing currently appears to be the most suitable rendering algorithm for computing predictive results in the future.

### 8.2.4 Real-Time Evaluation of Precomputed Data

As the previous sections showed, interactive computation of predictive global illumination solutions including complex geometry, light, and material is not possible today. Since there exist several application areas that rely on rendering such high-quality solutions interactively, researchers proposed to interactively display precomputed illumination solutions that contain all relevant effects.

Due to the wide area of possible applications ranging from lighting design over material checking to interactive television and games, a vast amount of approaches was developed. Complete coverage of all these approaches is out of the scope of this thesis, which concentrates on interactive, predictive rendering in Virtual Reality. One of the fundamental properties of most VR systems is the ability to navigate freely in the scene. Therefore, image relighting techniques [344] which require the viewer to keep

a fixed position and view direction, and approaches like Quick Time VR [60] will not be covered. In addition, this thesis assumes a standard scene composition consisting of geometry, materials, and lights, since this approach allows for close inspection of geometries without noticeable artifacts. Therefore, image-based techniques [481] that omit geometry explicitly (e.g., light-fields) and therefore feature a limited geometric resolution will not be covered as well. Finally, impostor techniques for efficient rendering reducing geometric detail at a macroscopic level were covered in Chapter 5 already.

### Surface Light Fields

Having a scene consisting of geometric objects, materials, and light sources at hand, interactive predictive rendering can be done in a straightforward manner: In a preprocessing step (partial) solutions to the Rendering Equation 8.1 are precomputed for a number of points on the surfaces of geometric objects and a set of view directions. To handle the huge amount of precomputed data, compression techniques are applied. Then, parts of these solutions matching the current view settings are displayed.

Such an approach was first applied by Miller et al. [363]. The authors precompute the 4D function, which they call a Surface Light Field (SLF), as a set of view-dependent textures which are compressed by a block-wise scheme similar to the one used for JPEG [246] compression. Unfortunately, this scheme prohibits random access decompression, which is essential for efficient rendering, especially on existing GPUs.

This problem was resolved applying other compression schemes. Wood et al. [586] suggested compression based on PCA or vector quantization of *lumispheres* (i.e., 2D functions defined at each surface point which specify view-dependent colors). Chen et al. [62] factorized the 4D function into spatially dependent and view-dependent parts using PCA, non-negative matrix factorization, or non-linear optimization [219] followed by vector quantization and lossy texture compression. Subr et al. [503] experimented with memory efficient low-frequency spherical harmonics (SH) representations of lumispheres. Kitahara et al. [269] proposed a progressive compression and reconstruction scheme that applies wavelet transform to PCA components and transmits resulting data in an importance based order to allow for more efficient remote rendering.

SLF acquisition, compression, and rendering was also extended to handle view-dependent opacity values [535] which enables much coarser base geometry. This is especially important if SLF data is acquired from existing scenes. In order to handle large scenes, Coombe et al. [79] proposed an efficient method for incremental construction and compression of SLF data based on incremental PCA [46]. SLFs were also applied to point-based rendering [503].

Summarizing findings from existing publications, SLFs represent a straightforward

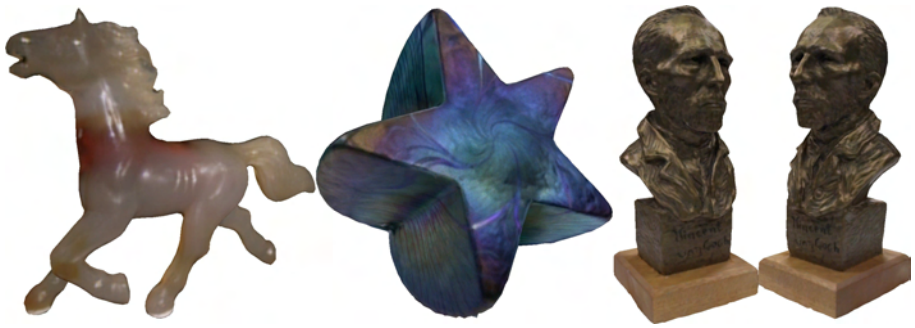


**Fig. 8.1: Images rendered from SLFs using the method of Chen et al. [62].**

way for precomputing global illumination for fixed scenes. Good results can be achieved in a number of scenarios (cf. Figure 8.1). Nevertheless, several drawbacks remain.

- First, SLFs consume large amounts of memory. The compression techniques proposed so far significantly reduce this amount of memory but especially for large scale objects or scenes, memory requirements remain very high. Therefore, better techniques need to be devised. An example of such an improved techniques is detailed in Chapter 10.

- Second, while SLFs are capable of capturing scenes with high-frequency lighting, handling of combinations of high-frequency lighting and glossy-specular materials requires tremendous amounts of memory. Therefore, practical applications limit themselves to glossy-diffuse materials.

- Third, precomputation of accurate SLFs for virtual scenes requires very long computation times. Since SLFs are fairly inflexible – geometry, material, and lighting are fixed – this prohibits use of SLF rendering for interior design where lighting or materials need to be changed. This can be improved by the technique presented in Chapter 10 to some extent. More flexible techniques are presented in the following.

### Object BTFs

To improve upon the flexibility of precomputed illumination solutions, precomputation of object (or scene) appearance for varying view and light directions was proposed. Due to the similarity with material BTFs (cf. Chapter 3) the resulting data will be called an *object BTF* in the following. As in the case of SLFs, the amount of precomputed data is huge and thus needs to be reduced. As for material BTFs, existing methods compress this data assuming either view- and light-dependent textures [391, 155], or spatially varying apparent BRDFs [303, 368].

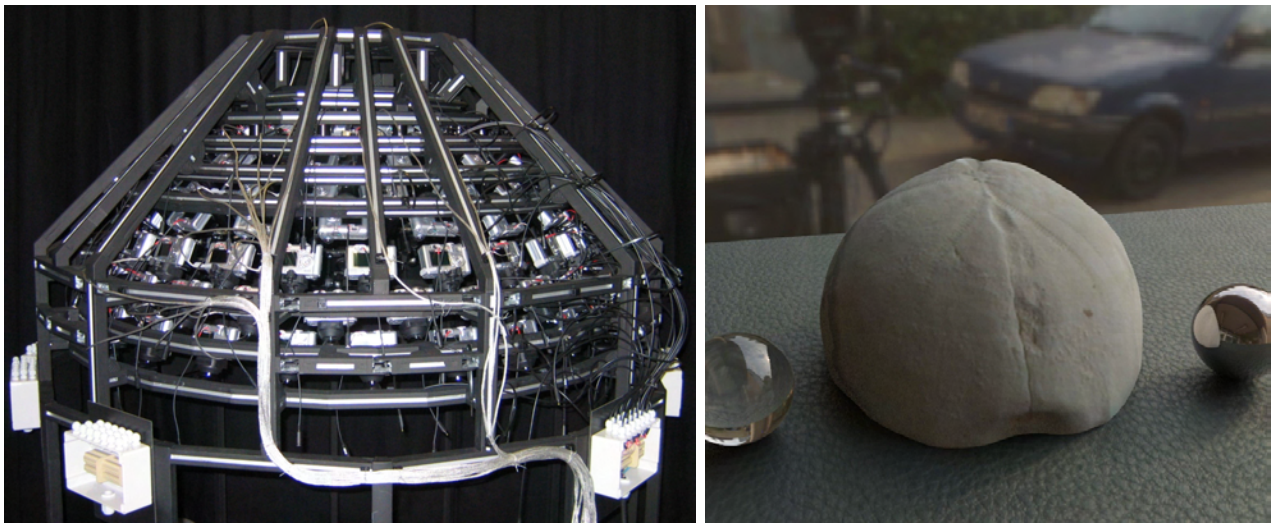


**Fig. 8.2: Object BTF [368]. Left: Measurement device consisting of 151 digital cameras positioned on the hemisphere above a measured probe. Right: Image of an acquired echinite object BTF in a virtual environment.**

Since precomputation of object BTFs requires even more time than precomputation of SLFs, existing techniques handle data acquired from existing objects. Nishino et al. [391] perform automatic data acquisition using a goniometer-like setup (cf. Chapter 2) and apply PCA-based compression to the per-face textures. Furukawa et al. [155] use a similar acquisition setup but improve upon the compression ratio using tensor product expansion, which allows for selective compression of dimensions of the BTF (e.g., view-directions only) but is less efficient to decompress.

A first approach based on compression of spatially varying apparent BRDFs was presented by Lensch et al. [303]. They acquired objects with a limited number of BRDFs using a manual acquisition process driven by best-next-view planning. They compressed resulting data using Lafortune models [290]. Müller et al. [368] densely sampled the view and light directions with a multi-camera setup, giving rise to very fast acquisition. They resample all data to a fixed set of local view and light directions and apply very efficient PCA compression of clusters of similar apparent BRDFs (cf. Figure 8.2 for results).

An approach also taking into account volumetric subsurface scattering effects was presented by Gösele et al. [165] but their approach is limited to diffuse surface materials.

Compared to SLFs, object BTFs are more flexible since they allow for changing lighting situations. This flexibility leads to the problem that evaluating their appearance with complex lighting situations again requires costly raytracing. Additionally, the basic problems of SLFs (large memory requirements and high precomputation time) are the same for object BTFs, limiting them to very special applications.

### Environmental Lighting

Already very early in the history of computer graphics researchers thought about efficient implementations of shading effects due to complex lighting. In 1976, Blinn and Newell [37] proposed environment maps as an efficient method for computing perfect reflections. The technique assumes that the size of an object is small compared to the distance to the lighting environment which allows the assumption that the amount of incoming radiance depends on the direction only (i.e., spatial dependence can be neglected).

Techniques for environmental lighting (also called *image-based lighting*) were later extended to support more complex materials. Miller and Hoffman [364] showed that environmental lighting of objects covered with diffuse or isotropic glossy BRDFs can be achieved by prefiltering environment maps with BRDF-dependent kernels. In their work, they successfully presented kernels for the diffuse and the specular part of the Phong model. Heidrich and Seidel [217] introduced a filter kernel for the anisotropic, glossy Banks [19] BRDF model, and showed successful combinations with normal mapped geometry. Additionally, they proposed prefiltered environment maps to simulate glossy transmission effects.

A generalization to arbitrary isotropic BRDFs was presented by Kautz and McCool [259]. They suggested to approximate BRDFs by a sum of radially symmetric lobes, one for each sampled elevation angle of the view direction. Image-based lighting for this representation can be achieved by first prefiltering the environment map with the kernels corresponding to the lobes for the different elevation angles. To render from a given view direction, the view direction is reflected at the lobe's main axis and the elevation angle of the resulting vector is computed. Finally, a shading value is computed by trilinear interpolation w.r.t. the surface normal and the current elevation angle, which can efficiently be implemented using 3D texture hardware. The technique was applied to rendering BTFs modeled as per-texel BRDFs by McAllister et al. [348]. A technique based on this principle but handling BTFs compressed with the more accurate technique presented in Chapter 3 is described in Chapter 9.

Several other papers significantly contributed to the success of methods based on prefiltered environments: Kautz et al. [261] proposed to replace lengthy prefiltering operations by a fast, hierarchical method suitable for arbitrary BRDFs. Latta and Kolb [294] applied homomorphic factorization [349] to the 4D function resulting from convolving environmental lighting with an isotropic BRDF. The factorization into two 2D functions allows for extremely efficient evaluation on graphics hardware since these functions can be stored in textures.

Another direction of improvement was followed by Cabral et al. [53]: They aimed

at simulating close lighting environments by computing individual environment maps for each vertex of an object. The proposed method works on a set of environment maps acquired from different view points. For each vertex individualized lighting is computed by interpolating the environment maps corresponding to closest points from the acquisition phase using BRDF-dependent warping schemes. The authors presented real-time performance for relatively complex scenes. Yet, warping errors are to be expected for complex BRDFs and environments with non-trivial occlusion.

A great enhancement of methods based on prefiltered environments was based on the insight, that other bases for the incoming light than the directional one should be used. In concurrent work, Ramamoorthi and Hanrahan [431] and Basri and Jacobs [22] proved that almost accurate image-based lighting of diffuse materials can be achieved in a nine-dimensional linear subspace. They both proposed representing material and lighting environments by third order Spherical Harmonics (SH) [333]. Since Spherical Harmonics basis functions are orthonormal, and since they allow for efficient rotation, this gives rise to a very efficient rendering algorithm [431]. In a follow-up publication, Ramamoorthi and Hanrahan [432] extended this concept to SH lighting of arbitrary isotropic BRDFs.

Recently, Sloan et al. [489] applied a similar idea to efficiently render BTFs in distant lighting environments. They represent the BTF's light direction as zonal harmonics coefficients instead of employing a directional basis since zonal harmonics allow for even more efficient rotation than spherical harmonics. As a very desirable result of this approach, parametric BTFs can be handled (i.e., BTFs where the surface structure or reflectance depends on parameters).

The existing methods for image-based lighting achieve very good results for a wide variety of applications since lighting environment can usually be assumed to be distant. Nevertheless, these methods are not capable of modeling important effects like self-shadowing of objects or self-interreflections (e.g., when computing reflections for a concave object the object itself will never be visible in the reflections). Additionally, interaction with non-distant elements like local lights or other geometries is very difficult. A straightforward approach based on dynamic environment maps was presented by Kautz [256] but it is limited to relatively simple scenes with a small number of objects.

More information concerning methods for environmental lighting, including in-depth descriptions and a comparison of existing methods, can be found in the surveys by Heidrich [213] and Kautz [255].

### Precomputed Radiance Transfer

The previous two sections presented different methods for computing the appearance of objects with complex materials in complex lighting environments. On the one hand, objects BTFs efficiently store the amount of outgoing radiance (including self-shadowing and self-interreflections) for a pair of view- and light-direction but are not suitable for efficient rendering given complex light situations. On the other hand, methods for image-based lighting efficiently handle complex lighting but lack support for local effects.

In 2002, Sloan et al. [487] published a seminal paper joining these two directions leading to a method called *precomputed radiance transfer* (PRT). At the heart of the method is the idea of transferred radiance, i.e., the amount of radiance received by some surface point. Having this quantity at hand, which includes contributions from multiply scattered or reflected light, the outgoing radiance can easily be computed if the BRDF of the surface point is known (cf. Figure 8.3). This computation additionally becomes very efficient if suitable representations for the BRDF and the incoming radiance are chosen (e.g., spherical harmonics).

Sloan et al. proposed to compute transferred radiance by multiplying incoming radiance from a lighting environment, represented by a number of SH coefficients as in [431], by a transfer matrix, which accounts for self-occlusion and inter-reflections. If only self-occlusion is considered, this transfer matrix can, e.g., be determined with a binary hemi-cube method that encodes occluded and unoccluded parts of the surface point's hemisphere. Interreflections can be included using raytracing in the preprocessing step.

The authors propose an especially efficient method for diffuse BRDFs and a less efficient method for arbitrary isotropic BRDFs. They additionally show that shadows of objects can efficiently be computed for arbitrary distant lighting by precomputing transfer vectors at positions corresponding to grid cells surrounding the objects. Like other methods handling image-based lighting, PRT can profit from interpolating lighting environments captured at different positions in space. Nevertheless, many problems remained which were targeted by following publications.

One direction of improvement is the reduction of rendering time for arbitrary BRDFs. Lehtinen and Kautz [300] proposed to avoid evaluation of the costly product of the BRDF matrix, the transfer matrix, and the vector encoding distant illumination since only few components of the resulting vector are used effectively. As a first idea, they suggest to compute transfer matrices that compute outgoing radiance, i.e., which include BRDF or BSSRDF effects already. Second, they suggest to represent outgoing directions w.r.t. a basis of piecewise bilinear functions. Representations of specific directions in this basis results in at most four coefficients having non-zero values, which

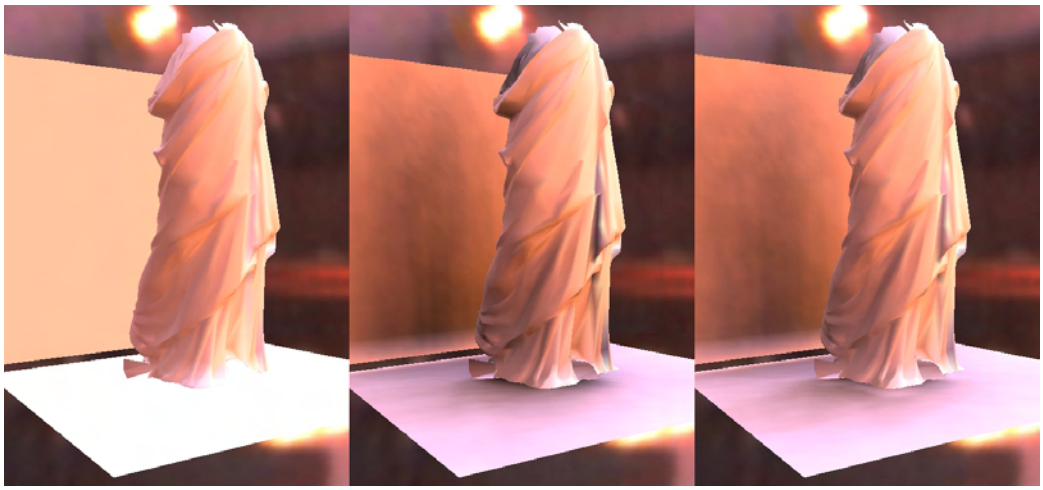


**Fig. 8.3: Low-frequency PRT rendering of a statue. Left: diffuse BRDF, center: including shadows, right: additional interreflection.**

reduces the evaluation cost from about $n^4$ operations to about $4n^2$ ($n$ being the SH order). Additionally compressing the transfer matrices with PCA allows to precompute the products of lighting and eigen-transfer-matrices on the CPU and thus leaves about $4k$ operations to be performed on the GPU per vertex ($k$ being the number of principal components). A similar solution was found in concurrent work by Sloan et al. [486] but they employ local PCA [251] to compress transfer matrices, which improves reconstruction quality using the same number of principal components.

Another direction of improvement concerns the accuracy when projecting hemispherical functions like BRDFs into spherical harmonics. To avoid influences of the lower hemisphere, Sloan et al. [486] proposed fitting SH of a fixed order to a given hemispherical function using nonlinear optimization. Gautron et al. [159] proposed using a basis derived from shifted associated Legendre polynomials combined in the same way as SH but covering the upper hemisphere only. To enable computations involving both hemispherical and spherical data (e.g., a BRDF and incident, distant radiance) SH coefficients are converted into the hemispherical basis by sparse matrix multiplication.

A third, larger area of improvements is concerned with removing the restrictions of PRT to low-frequency lighting environments. Ng et al. [382] were the first to point out that SH are less suitable for representing lighting environments also containing important high-frequency information than other bases. As a result, they proposed to project environmental lighting into a Haar wavelet basis. To reduce the large number of wavelet coefficients, they perform non-linear compression (i.e., they set the weights of wavelets with minimal contribution to zero), which leads to a better preservation of all-frequency lighting effects with the same number of basis functions than if using SH. Unfortunately, their method for non-linear compression later requires sparse matrix

multiplication which cannot be implemented efficiently on the GPU so far. In addition, BRDFs need to be parameterized w.r.t. a global coordinate system since Haar wavelet coefficients cannot be rotated efficiently. As a result, the authors were able to show interactive renderings for versions with reduced dimensionality (e.g., diffuse BRDFs or fixed view-directions) only.

These restrictions were relaxed by Wang et al. [560] and Liu et al. [320] who factorize the BRDF into view- and light-dependent 2D functions [258] and integrate the light-dependent parts into the transfer matrix. Liu et al. [320] additionally compress the transfer matrices by local PCA. For rendering, the distant incident radiance is modulated by the transfer matrix – intermediate results can be reused if the lighting remains fixed – and the final shading value is reconstructed by a few (e.g., 4) dot products of view-dependent BRDF parts and transferred radiance.

Recently, Green et al. [174] proposed a variant of PRT based on prefiltered environment maps that enables efficient rendering of high-frequency view-dependent effects like glossy reflections. The authors approximate the transfer functions per vertex and view-direction, which are thus 2D functions, by a sum of a fixed number of 2D Gaussians. If this approximation is done in a consistent way, interpolation of transfer functions w.r.t. position and view-direction reduces to interpolation of parameters of 2D Gaussians, which enables accurate glossy reflections. For efficient rendering the environment map is prefiltered with Gaussians of varying size. While the approach achieves real-time frame rates for effectively simulating view-dependent effects like reflections in glossy BRDFs, it should be combined with other approaches to compute view-independent effects. A severe problem of the method is the necessity to enforce consistency of fitted Gaussians, since interpolation from inconsistent Gaussians produces severe artifacts. While being difficult in most cases already, this problem might be impossible to solve if transfer matrices vary significantly, e.g., due to complex occlusion.

A similar but more general approach was recently proposed by Tsai and Shih [523]. Their implementation approximates both lighting and transfer functions by spherical radial basis functions similar to the ones used by Green et al. While decomposing lighting and transfer into optimally chosen spherical radial basis functions requires longer than using (more restrictive) SH or Haar wavelet decomposition schemes, they yield accurate function representations for a smaller number of basis functions. This leads to faster rendering times when applying suitable BRDF factorization scheme [560, 320]. Compared to Haar wavelets, spherical radial basis function additionally allow for efficient analytic rotation.

A very interesting and efficient method for interactive rendering using high-frequency lighting was published by Overbeck et al. [401]. In the first frame they use all Haar

wavelet coefficients to render a very accurate image. In following frames, the lighting changes w.r.t. last frame are encoded as a small number of Haar wavelets which incrementally correct the existing solution. This approach allows convergence against an optimal solution. In addition, by carefully avoiding ghosting artifacts, also in-between frames are of very good quality.

A more flexible PRT approach was presented by Ng et al. [383]. It handles BRDFs, lighting, and visibility independently and thus enables fast replacement of materials in the scene at the cost of omitting inter-reflections. The authors project all three ingredients into a Haar wavelet basis and thus need to evaluate triple products to evaluate the Rendering Equation. Unfortunately, these do not yield as simple solutions as double products of orthogonal basis functions. Nevertheless, they present an efficient evaluation algorithm for the Haar wavelet basis, especially if non-linear approximation is applied. Still, some seconds are required for rendering a single frame.

A fourth group of improvement methods tries to apply PRT to a wider range of materials (cf. Figure 8.4). Kautz et al. [260] enabled rendering of arbitrary anisotropic BRDFs by representing BRDFs as matrices that get multiplied with the incident transferred radiance. From the resulting vector, the entries corresponding to closest view directions need to be selected and interpolated. This approach was extended to BTFs by Sloan et al. [488]. Since BTFs are represented as per-texel apparent BRDFs, extension of Kautz et al.'s method only requires per-fragment lookup of respective SH apparent BRDF coefficients. Storage requirements for this method are kept handleable since the authors employ BTFs with small spatial extent only. Later, Müller et al. [370] removed these memory restrictions by compressing both transfer matrices and the BTF using local PCA, which enabled the use of much larger BTFs. Their approach precomputes the transferred outgoing radiance for all combinations of eigen-transfer-matrices and eigen-ABRDFs of all clusters on the CPU and makes the results available to the GPU by uploading a texture. This operation is rather time-consuming and therefore limits rendering speed when rotating lighting environments. A similar approach combining BTF materials and all-frequency lighting is described in more detail in Chapter 10.

Another type of potentially interesting materials are those showing considerable subsurface scattering effects. Although these can be handled if the transfer matrices include material effects already, precomputation of such PRT data for all-frequency lighting requires extensive preprocessing. This overhead can be reduced significantly by the method of Wang et al. [559]. Finally, approximate inclusion of low-frequency effects resulting from wave-length dependent materials was recently presented by Lindsay and Agu [314].

Another severe problem of the original PRT method is missing interaction with other objects and local lights. Such problems were tackled by a fifth group of publications.

**Fig. 8.4: Extending PRT to more general materials. Left: diffuse BRDF, center: Phong BRDF, right: plasterboard BTF.**

Annen et al. [7] aimed at removing the necessity of sampling incoming radiance at a number of points in space for simulation of close objects and local lights. They proposed to compute incident radiance at arbitrary points in space from the SH coefficients of an environment map captured at the center of the object and SH coefficient gradients, which corresponds to a first order Taylor approximation. Good results rendered interactively are shown in their paper and even more convincing results can be achieved when updating coefficients and gradients dynamically [256]. Nevertheless, the approach will fail in environments with complex occlusion. An approach similar to the one of Annen et al. but based on interactively changing illumination coefficients to mimic zooms onto suitable regions of the hemisphere was proposed by Wang et al. [555].

Specifically targeted at lighting design, Kristensen et al. [282] subdivide the space where light sources might possibly be placed by a grid and place light sources at grid vertices. They precompute exitant transferred radiance per vertex and light source, project the results into SH, and apply clustered PCA to compress this data. To reduce the large number of lights, light sources that lead to similar outgoing radiance are clustered. To avoid management of clusters per vertex, the authors define regions of vertices with identical clusters, which should be fairly difficult in environments with complex occlusion. Nevertheless, interactive results for a reasonably complex environment from architecture are presented.

Very recently, Kontkanen et al. [277] presented an approach supporting close-range illumination at the cost of restricting the specularity of materials in the scene. The authors represent direct lighting in a wavelet basis and define a transport operator transferring incident to bounced incident light. Multiple applications of this operator yield a global light transport operator similar in style to the radiosity approach. A similar technique supporting arbitrary materials but requiring fixed views was recently published by Hasan et al. [204].

More complex interactions of close objects are possibly with the method of Mei et al. [353]. They improve rendering of shadows from and interreflections between nearby

objects. To achieve this goal, they render potential objects into textures for a set of fixed directions and store either occlusion information or locations of visible points and reflection directions. For interactive rendering of shadows and interreflections between objects, distant lighting environments are represented by a set of (clustered) directional lights. Rendering itself resembles raytracing but utilizes precomputed information. The authors were able to document interactive to real-time performance for simple scenes with few objects. Unfortunately, it seems questionable if this approach scales well with the number of close objects in a scene.

Even more complex interactions become possible if deformable objects are allowed. James and Fatahalian [236] approximated PRT for deformable objects by taking a series of model states from physics-based simulations (each referring to a single force applied to the model) and precomputing transfer matrices for these. They linked model states to PRT transfer matrices and reduced the dimensionality by applying SVD. For interactive rendering, they inspect the model's state and reconstruct respective transfer matrices from the SVD representation. While this approach works nicely for a very limited number of model deformations, a huge set of input model states would be necessary for arbitrary deformations, which would increase precomputation efforts and storage requirements immensely.

Therefore, Kautz et al. [257] presented an approach relying on online recomputations. The authors suggest to recompute transfer matrices for every vertex whenever necessary. Typically, this is accomplished using the hemi-cube method [72] if interreflections are neglected. Unfortunately, this requires rendering surrounding geometry five times. Therefore, Kautz et al. proposed to project surrounding geometry onto the vertex' tangent plane after projecting it onto the vertex' hemisphere, and to store resulting data in a low-resolution 2D array. The approach obviously requires geometry to be rendered only once. Additional speed-ups can be achieved by employing multiresolution geometry. Interactive results were presented for scenes with a rather small number of vertices and it seems questionable how well this method will scale in more complex environments.

Finally, a number of limitations of existing PRT methods need to be resolved when applying PRT to industrial data. Dmitriev et al. [110] pointed out shading problems due to badly shaped triangles and cracks in the geometry, which often result from tessellating NURBS model. Other problems include shadow leaks due to slightly overlapping triangles. Resulting errors can be fixed by repairing geometry, adjusting triangles, and determining visibility and interreflections for samples distributed over the whole model (instead of at vertices only). Additional problems include extensive preprocessing times and memory requirements for industrial-sized models.

### Discussion

In summary, PRT methods nicely simulate complex lighting of objects with complex materials while allowing interactive changes to the lighting environment. Therefore, they represent suitable solutions for many Virtual Reality applications like interior designs already.

Nevertheless, all methods need to trade off preprocessing time, storage requirements, and evaluation time. Therefore, some methods are more suitable for rendering specific effects than others. Typically, good results are achieved for either low-resolution materials or low-resolution lighting environments. Therefore, it appears suitable to add effects from high-frequency materials and high-frequency lighting with different methods, especially if non-distant effects are considered. Such an approach was published by Dmitriev et al. [110]: While most globally illuminated objects are rendered using low-frequency PRT, reflections in a nearly perfect reflector are simulated by a path tracer.

Additionally, PRT methods still feature severe limitations on geometry or material modifications. Therefore, future research, which might possibly lead to an inclusion of precomputed data in ray-tracing frameworks, will need to resolve these limitations to allow for more general areas of application.

## 8.2.5 Real-Time Computations for Specular Materials

As shown above, methods for rendering global illumination solutions from precomputed data are very successful at handling most types of scenes. Unfortunately, they lack the ability to capture interactions of high-frequency lighting and high-frequency (i.e., glossy or specular, potentially transparent) materials efficiently. If physically accurate rendering is desired, these interactions nevertheless need to be simulated somehow.

Therefore, this section presents results from existing work concentrating on computing these difficult interactions in real-time. Resulting effects can roughly be categorized into three classes:

- refractions (i.e., images of geometry seen through transparent objects),
- reflections (i.e., images of geometry on reflecting surfaces), and
- caustics (i.e., regions lit brightly due to focused reflected or refracted light).

Computation of each of these three effects is a demanding task, especially if generality is required (e.g., recursive reflections in opposing mirrors). Therefore, interactive algorithms typically make simplifying assumptions, usually handling only one type of effect. In the following, existing methods are therefore grouped into these categories.

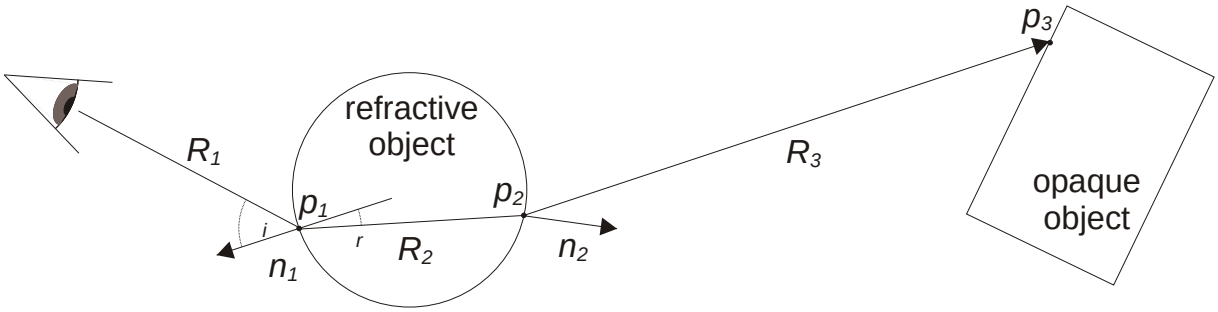



**Fig. 8.5: Basic setting for computing refractions.**

This section covers interactive methods only. Non-interactive, highly accurate methods that also handle interactions with glossy or specular materials are described in Section 8.1 already. Additionally, no description of methods based on (distributed) ray or photon tracing is given, since they were presented in Section 8.2.3 already. Finally, methods for rendering translucent objects, i.e., objects showing significant subsurface-scattering behavior, are not covered here, since translucency smoothes out high frequencies. Therefore, such objects can efficiently be handled using methods based on precomputed data. Interested readers find links to methods for interactive rendering of translucent objects in the recent publication of Haber et al. [192].

## Refractions

Refraction effects are seldomly considered in interactive renderings. Although such effects can commonly be observed in every type of glass surfaces, they can safely be omitted as long as the object is relatively thin, its thickness does not vary, and the object is almost perpendicular to the viewer. Yet, for all other transparent objects like curved glasses, lenses, or glass bowls, simulation of refraction is absolutely necessary.

The basic approach for computing refractions is very simple (cf. Figure 8.5): A view ray $R_1$ hits a refracting object at point $p_1$. A refracted ray $R_2$ is computed based on Snell's law $m_i \sin(\theta_i) = m_r \sin(\theta_r)$, where $m_r$ and $m_i$ denote the refraction coefficients of the refractive object and its surrounding, and $\theta_i$ and $\theta_r$ denote the angles between $R_1$ respectively $R_2$ and the normal $n_1$ in point $p_1$. Next, the exit point $p_2$ and its normal $n_2$ are determined and the exit ray $R_3$ is computed. The displayed color is finally computed by determining and shading $p_3$. Unfortunately, an exact implementation of this scheme requires raytracing, which is hard to implement as an interactive tool.

A first, simplistic approach for interactive refraction rendering was presented by Oliveira [397] (cf. Figure 8.6). Initially, the opaque objects are rendered into a texture. Then, for every vertex of the transparent objects a refracted ray $R_2$ is computed. Next, this ray is intersected with the textured plane and the texture value corresponding to

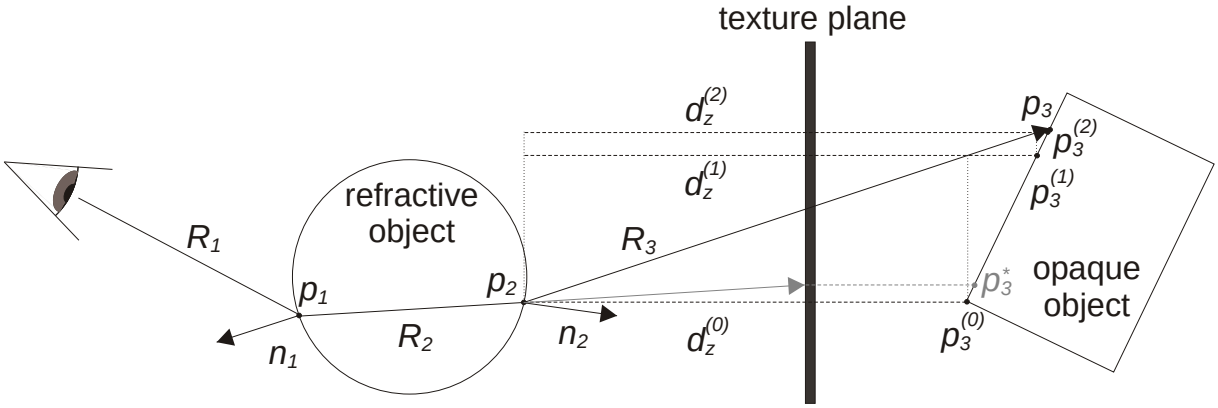



**Fig. 8.6: Solutions schemes for refractions. The approach of Oliveira [397] computes $p_3^*$ by intersecting $R_2$ with the texture plane (gray). Ohbuchi determines a much more accurate point $p_3$ by an iterative depth determination scheme [395] (black).**

the intersection point is rendered. Implementations of this approach on programmable hardware achieve real-time framerates for fairly complex objects [494]. Even faster results can be achieved by indexing environment maps based on the refracted view direction [313]. Unfortunately, parallax effects due to complex geometries and multiple refractions are not possible.

An approach showing approximate parallax effects, handling double refractions, and multiple refractive objects was presented by Ohbuchi [395]. In a first step, opaque objects are rendered into a texture, retaining depth information. Then, for each vertex of the transparent object, a refracted ray is traced through the transparent object on the CPU and the position and direction of the ray leaving the object are stored. In the final step, the intersection of this ray $R_3$ and the scene geometry in the depth enhanced texture are computed, and the texture coordinate of the intersection point is assigned to the vertex. To compute the intersection point, it suffices to compute the distance between exit and intersection point. The method employs a clever, interactive method to determine this value (cf. Figure 8.6). First, a depth value $d_z$ is initialized to zero. Then, $d_z$ is improved iteratively in the $i$-th step by looking up a new depth value $d_z^{(i)}$ and setting $d_z$ to $d_z^{(i)}$. As long as the geometry in the scene is smooth, this approach will converge against the desired depth value.

To remove the performance limitations associated with tracing rays through the refractive object per vertex on the CPU, Wyman [593] proposed an algorithm that efficiently approximates the parameters of the exit ray $R_3$ (cf. left image in Figure 8.7). Since it runs entirely on the GPU, real-time performance is achieved even for relatively complex, transparent objects. The algorithm starts with a vertex program that computes the refracted ray from the vertex position and refracted view direction. The location of $p_2$, the approximate exit point, is computed by interpolating the depth values $d_n$ and

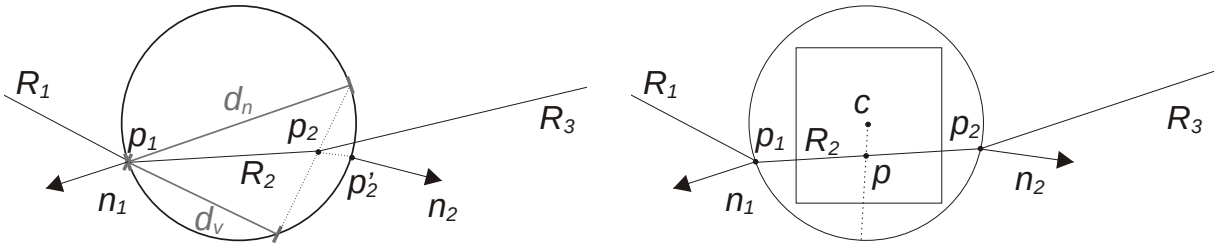



**Fig. 8.7: Methods to determinate twice refracted ray. Left: Wyman's approach [594], right Chan and Wang's approach [59].**

$d_v$ in a reasonable way. To determine the direction of the approximate exit ray $R_3$ the normal at $p_2$ is required, which is determined by indexing a texture containing the normals of the backfacing triangles. Obviously, since the parameters of the exit ray are approximated only, arbitrarily large errors can be seen in the final image. Additionally, errors are introduced by handling exactly two refraction points, which prohibits correct handling of non-convex objects. Nevertheless, for relatively convex objects, reasonable results are achieved. The approach can efficiently be combined [594] with methods capable of approximating parallax effects (e.g., the one from Ohbuchi [395] described above or the ones from Heidrich et al. [215] and Yu and McMillan [606], which represent surrounding geometry as light field).

Another method for computing exit rays was presented by Chan and Wang [59] (cf. right part of Figure 8.7). They approximate each transparent object by a set of textured cubes. Each cube map encodes the distance to the object surface for the sampled directions. Given the start position $p_1$ of the refracted ray $R_2$, this cube map now helps to determine the exit point $p_2$ in the following way. A point $p$ is advanced from $p_1$ along the refracted ray direction until its distance to the cube's center $c$ matches the depth value stored in the textured cube for the direction $p - c$. With the help of an intelligent, adaptive step-size control for advancing $p$, interactive frame rates for models with several 10k vertices can be achieved. Unfortunately, it is very difficult to approximate arbitrary objects with cubes, leading to severe artifacts for concave objects.

## Reflections

Considering the basic problem of tracing rays, computing reflections is even more simple than handling refractions: For each view ray hitting a reflective surface, the ray is mirrored around the normal and traced through the scene. Yet, unlike refracted rays, which typically follow a similar direction as the original ray, reflected rays typically feature a largely different direction. In raytracing applications, this leads to the problem of almost random memory access patterns during ray traversal, making it even more difficult to achieve interactive performance than without considering reflections.

For the special case of planar surfaces, accurate and efficient results can be achieved without raytracing by mirroring the view-point in the surface plane and rendering an image of the scene from the mirrored point. Multiple, recursive reflections can be simulated using multipass rendering [105]. Also, the approach can be extended to handle glossy reflections using hardware accelerated computation of summed-area tables [218]. Unfortunately, all these approach require that the full scene be rendered once for each planar reflector, which might be too slow for complex scenes.

To resolve this problem, Bastos et al. [23] proposed an image-based approach that first renders the scene into an environment map with depth information. For efficient rendering, the points in the environment map are warped to the view from the mirror point (which requires computation time proportional to the number of pixels in the environment map). To avoid disocclusion artifacts due to warping, multiple points can be stored for each pixel of the environment map. Unfortunately, forward mapping cannot efficiently be implemented on existing graphics hardware.

An interesting application of rendering planar reflections including refraction effects and polarization was demonstrated by Guy and Soler [190]: They map raytracing of cut gemstones consisting of planar facets to graphics hardware and achieve real-time performance. Unfortunately, interactive rendering for models with more than a couple thousand polygons seems hardly possible.

While rather efficient implementations for rendering reflections in planar surfaces exist, general solutions for curved surfaces are much more difficult to realize. If the reflecting object is small compared to the distance to neighboring objects, methods based on prefiltered environment maps can be used. Naturally, the approach can be combined with methods handling planar reflections [389]. Additionally, methods handling non-distant objects exist. Heidrich et al. [215] and Yu et al. [606] both proposed to represent the environment of a reflecting object as a 4D light field, which allows them to render parallax effects at the cost of largely increased memory requirements and limited sampling rate. Szirmay-Kalos et al. [508] applied an iterative solver based on the secant method similar to the one of Ohbuchi [395] to render non-distant objects. They achieve real-time results for complex reflecting objects and additionally handle refractions and caustics. Nevertheless, all these approaches are not viable if objects are positioned very close to the reflector.

Recently, Popescu et al. [418] proposed an approach supporting close objects. For every reflecting object, potentially reflected objects are approximated by a planar impostor storing per-pixel color and depth information. Then, in a fragment shader, the reflected ray is intersected with each of the planar impostors, taking into account the per-pixel depth information. While the approach handles small but close objects relatively well, it is neither suitable for high-resolution reflections nor large, close objects

that potentially surround the reflector.

A different method achieving very accurate results was proposed by Ofek and Rappoport [394] which handles arbitrary parallax effects by computing virtual vertices of mirrored objects on the CPU. An object potentially mirrored in a curved reflector is subdivided as finely as necessary, the mirrored vertices are computed, and finally a mesh with the mirrored vertices is rendered. An important speedup for computing mirror vertices is achieved by the explosion map data structure, which determines efficiently which vertices are mirrored in which reflector triangles. Unfortunately, correct handling of non-convex surfaces is impossible and the required subdivision techniques are ill suited for GPU implementation, making the approach less attractive for current hardware configurations.

In Chapter 11 two methods for computing reflections in curved surfaces similar to the approach of Ofek and Rappoport but running completely on the GPU are described. While the first handles Bézier surfaces, the second targets polygonal meshes. Since such approaches cannot accurately handle concave surfaces, another method based on GPU raytracing is described which yields accurate results for arbitrary reflector geometries.

### Caustics

The third important visual effect caused by specular surfaces are caustics. Caustics are bright regions on a diffuse surface due to light getting bundled because of reflecting or refracting objects. While caustics appear in many situation, early methods concentrated on caustics caused by water surfaces, e.g., in swimming pools or in the sea.

The first method achieving believable results at real-time performance was presented by Stam [496]. He proposed precomputation of tileable caustic textures caused by water surfaces. Several textures for varying depth and wave patterns were computed and Stam paid special attention to coherency to enable blending between different depths and time steps, which enabled believable animations. Naturally, rendering these textures at runtime does not produce accurate results since the caustics do not match the actual wave patterns at the surface.

With the advent of programmable GPUs, Trendall and Stewart [522] proposed a framework for computing caustics on the GPU in real-time. They approximate the equation for computing caustics from height field water surfaces by a double sum of 1D convolutions which can efficiently be evaluated using the GPU image processing pipeline. They achieved interactive frame rates for complex height fields even on then modern GPUs. Yet, their method assumes an approximately planar height field for describing the water surface and a planar caustic receiver surface, making it ill suited for ocean scenarios. In addition, the do not consider shadow casting objects in the water.

A less performant but much more general and realistic approach was presented by Iwasaki et al. [233]. They subdivide the water surface into triangles and the water volume into illumination volumes. For each water surface triangle, one illumination volume is defined by computing the refracted light directions at the three vertices. These volumes are further subdivided into tetrahedral volumes such that bilinear interpolation of the illumination values at tetrahedra vertices gives an accurate estimate of the illumination values at interior points. The volumetric representation enables rendering of light shafts by accumulating direct illumination and scattered light. Caustics on arbitrary surfaces can be computed by intersecting illumination volumes and surface geometry on the CPU and rendering results on the GPU. The approach additionally handles shadow effects due to objects in the water and provides reasonably good results at interactive rates. One drawback of the method is the extensive computation requirement for computing illumination volumes and tetrahedral subvolumes, which limits the complexity of the water surface. Another problem, i.e., costly determination of caustics, was improved in a following publication by Iwasaki et al. [262]. The authors speed up intersection computations by slice-based volume rendering techniques. This way, they are additionally able to approximately render objects reflected or refracted in the water surface. Unfortunately, the approach remains computationally expensive if a significant number of caustics receiver objects are present in the scene. A faster caustic renderer based on illumination volumes completely running on the GPU was later proposed by Ernst et al. [129]. Like previous methods, it handles direct caustics only. Unlike previous methods, caustics my be caused by arbitrary objects but incorporation of shadows is not possible. Partially due to implementation on the GPU (including specific optimizations) and partially due to performance improvements of GPUs, the approach achieves higher framerates than the approaches of Iwasaki et al.

Other approaches rendering caustics from arbitrary objects employ precomputed data. Wyman et al. [596] approximate the 8D caustic casting function of an object by a 5D function. The approximate function takes into account light direction and position of the caustic receiver but assumes diffuse receiver surfaces and omits the distance between caustic caster and light source. Interactive rendering becomes possible by interpolating sampled data from the 5D function. Due to the huge memory requirements for the precomputed data, the authors employ a raytracer running on a CPU cluster to render the data interactively. To compress the data, light directions can be represented in a SH basis, which makes rendering on existing GPUs possible at the price of reducing reconstruction quality significantly. Due to the lengthy precomputation step, objects cannot be deformed and interpolation artifacts caused by the high-frequency behavior of the sampled 5D function are well visible.

A similar strategy for caustics caused by refracted light was followed by Iwasaki et al.

[234]. They precompute light transport tables which store – for each vertex and sampled direction – the parameters of the ray leaving the object after being refracted internally. For rendering, they compute the incoming light direction for each vertex, create illumination volumes from bilinear interpolation of corresponding stored exit ray parameters, and render caustics with their volumetric rendering method [262]. Unfortunately, bilinear interpolation may introduce arbitrary errors for non-convex objects. In a following publication, Iwasaki et al. [235] propose computation of caustics by intersecting illumination volumes with bounding boxes instead of slicing planes, followed by projecting result onto the object surface. While this approach leads to view-independent caustics, making the approach more efficient and realistic, results become much worse if the bounding box represents a bad approximation of the object's shape.

A completely different technique for rendering caustics was proposed by Wand and Straßer [552]. They choose sample points on caustic casting objects such that each of these acts as a mirror, projecting the distant environment onto the caustic receiving surface. To render caustic surfaces, the radiance reflected from each such mirror point onto the currently processed surface point is computed by indexing an environment map, and summing contributions. Since this approach introduces significant aliasing, radiance is retrieved from regions of the environment map with the regions' shapes depending on the mirrors' curvatures. The algorithm achieves interactive results for fully dynamic scenes but is limited to a small amount of mirroring points, distant lighting, and direct caustics, and omits shadows.

Another technique for computing caustics is more closely related to Photon Mapping [239]: Vertices from caustic casting objects are splatted onto caustic receiving geometry, the idea being that vertices correspond to photons. Obviously, this approach requires a photon density estimation technique to generate caustics with low noise. Unfortunately, these techniques are known to be time consuming. Therefore, Shah and Pattanaik [469] propose application of a low-pass filter to a caustic texture which yields fast but inaccurate results. Wyman and Davis [595] instead propose density estimation in texture space, similar to the approach of Czuczor et al. [82]. While this approach might yield good results for well-parameterizable, angled surfaces, it may well produce very bad results, e.g., for spheres.

### Discussion

In summary, many methods for simulating light interactions with specular materials in real-time exist, most of them creating very realistic images for fairly complex scenes. Unfortunately, few of them handle more than one effect, and none of them is fast and general enough to be considered an ultimate solution. Therefore, significant research

will be required in the future to improve the current state of the art. Yet, it seems very likely that such effects are handled with raytracing techniques in the future since interactive rendering of such effects requires Whitted style shading [578] only.

### 8.2.6 Discussion

In the previous sections, a bunch of existing approaches for interactive rendering of global illumination solutions was presented. All of them feature specific advantages and disadvantages (cf. Table 8.1) making it impossible to determine an optimal solution for each and every application scenario. While caching methods clearly help to improve performance, they reduce rendering quality by introducing incorrect shading values, holes, update latencies, and ghosting artifacts. Methods based on radiosity allow for interactive walkthroughs of complex scenes but typically require substantial preprocessing time and fail to handle glossy or specular materials in an efficient way. Interactive raytracing methods represent the most general approach towards interactive global illumination rendering but suffer from large computational complexity, making them unattractive for todays standard graphics users. Methods based on precomputed data achieve predictive quality at very good frame rates at the cost of omitting high-frequency effects and requiring significant precomputation. Fortunately, high-frequency effects can be added with approaches targeting specular materials and arbitrary lighting.

Note, that the assessment of capabilities given in Table 8.1 is approximate only due to at least three reasons: First, a judgment of capabilities of a hole class of methods is given. Naturally, individual techniques show specific strengths and weaknesses on their own, making it difficult to judge all approaches in a unified way. Second, capabilities are judged on the basis of current technology and do not represent strict limitations of classes of methods. E.g., it comes hardly at a surprise that interactive raytracing methods still have problems handling glossy-diffuse materials, yet this might definitely change in the future. Third, the assessment is given in a relative scale only, since absolute judgments are not possible. Different application scenarios have different requirements, thus leading to a subjective ranking of techniques.

For application areas like walkthroughs or industrial design reviews, which are mainly targeted by this thesis, it is much more desirable to generate very accurate images at real-time frame rates than allowing geometries, lights, or materials to be changed interactively. For such cases, the currently best choice of technology appears to be a combination of methods operating on precomputed data and approaches handling specular effects. Especially important for powerwall or cave presentations, which require several high-resolution images being computed at the same time, such a combination generates interactive, accurate results on a single PC per displayed image.

| | dynamic scenes | | | | generality | | | | | present. | | resources | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\mathcal{A}$ | $\mathcal{B}$ | $\mathcal{C}$ | $\mathcal{D}$ | $\mathcal{E}$ | $\mathcal{F}$ | $\mathcal{G}$ | $\mathcal{H}$ | $\mathcal{I}$ | $\mathcal{J}$ | $\mathcal{K}$ | $\mathcal{L}$ | $\mathcal{M}$ | $\mathcal{N}$ | $\mathcal{O}$ |
| Ray Cach. | ± | ± | + | + | ± | + | + | – | – | –– | –– | - | ± | ± | ± |
| Samp. Cach. | + | ± | ± | ± | + | ± | + | – | + | – | –– | ± | + | + | + |
| 'Adv.' Rad. | – | –– | –– | + | ++ | –– | ++ | ++ | – | + | + | ± | ± | ++ | ++ |
| Rad. + RT | – | –– | –– | –– | ++ | ± | ++ | ± | –– | + | + | –– | –– | - | ± |
| Inst. Rad. | + | + | ± | + | ++ | – | ++ | + | - | + | ± | + | + | + | ++ |
| Rad. Cach. | ± | – | –– | –– | ++ | – | ± | ++ | ± | + | + | ± | –– | + | ++ |
| Clust. RT | + | ± | ++ | ++ | ± | ++ | + | ± | + | + | + | ± | ± | + | –– |
| Hardw. RT | + | – | ++ | ++ | – | ++ | + | ± | ± | + | + | ± | ± | ± | ± |
| GPU RT | ± | – | ++ | ++ | – | ++ | + | ± | ± | + | + | ± | ± | ± | ++ |
| SLFs | –– | –– | –– | –– | ++ | ± | ++ | ++ | – | + | + | –– | –– | ++ | ++ |
| Obj. BTFs | ++ | –– | –– | ++ | ++ | ± | ± | – | ± | + | + | –– | –– | ++ | ++ |
| IBL | ++ | ++ | + | ++ | + | + | –– | + | + | + | + | ± | + | ++ | ++ |
| PRT | ± | – | – | ++ | ++ | ± | ± | + | – | + | + | –– | –– | + | ++ |
| Spec. FX | + | + | ++ | ++ | –– | ++ | ± | – | + | + | + | + | + | + | ++ |

| dynamic scenes | | generality | | presentation | | resources | |
|---|---|---|---|---|---|---|---|
| $\mathcal{A}$ | rigid geo. changes | $\mathcal{E}$ | diffuse materials | $\mathcal{J}$ | image quality | $\mathcal{L}$ | memory |
| $\mathcal{B}$ | deformable geo. | $\mathcal{F}$ | specular materials | $\mathcal{K}$ | latency | $\mathcal{M}$ | precomp. time |
| $\mathcal{C}$ | material changes | $\mathcal{G}$ | close range fx | | | $\mathcal{N}$ | runtime |
| $\mathcal{D}$ | lighting changes | $\mathcal{H}$ | area lights | | | $\mathcal{O}$ | hardware |
| | | $\mathcal{I}$ | large scenes | | | | |

**Tab. 8.1: Comparison of properties of existing methods.**

In the following chapters, this thesis will present own contributions towards such a combination of rendering methods. In Chapter 9 a method for efficient rendering of scenes containing compressed BTF materials is presented. Lighting can either be represented by directional- or point lights, which are commonly supported by standard, real-time graphics APIs, goniometric lights, or environment maps. In order to capture precomputed illumination effects, Chapter 10 presents two methods. First, an efficient approach for generating and rendering Surface Light Fields. The approach achieves very high quality renderings for scenes of limited complexity and thus allows for effective design reviews. Second, a novel PRT approach, which also generates high-quality results and additionally allows for efficient rotation of distant lighting. Finally, Chapter 11 presents three approaches for interactively simulating reflections in curved specular surfaces. While the first two methods are based on projecting geometry onto reflecting surfaces (forward-mapping), the third method employs a real-time raytracer (backward-mapping) implemented on the GPU.

# Chapter 9

# Real-Time BTF Rendering

In the previous chapters different aspects contributing to rendering of realistic and potentially predictive images were described. Part II showed the increased realism due to accurate material representation. Chapter 8 motivated the use of accurate rendering techniques to compute images matching reality. In this chapter, efficient rendering algorithms for the BTF representation presented in Chapter 3 are proposed. The approaches support various types of light sources (point- and directional lights, goniometric lights, and environmental light). Additionally, ways for extending the methods in order to work with BTFs resulting from texture synthesis are presented. Since the rendering approaches implement local illumination models, they can easily be executed in real-time. Less efficient methods supporting global illumination effects are presented in the following chapters.

## 9.1 Problem Description

Rendering BTF approximations in real-time imposes a challenge both on the rendering method and the graphics hardware used for visualization. For every rendered surface point $\mathbf{x}$, which corresponds to a rasterized fragment, the reflection equation

$$L_o(\mathbf{x},\mathbf{v}) = \int_\Omega \mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) L_i(\mathbf{x},\mathbf{l}) \Big(\mathbf{n}(\mathbf{x}) \cdot \mathbf{l}\Big)\, d\mathbf{l}, \tag{9.1}$$

with outgoing radiance $L_o$, view direction $\mathbf{v}$, incident radiance $L_i$, hemisphere $\Omega$, and normal vector $\mathbf{n}$, has to be evaluated. If measured BTFs are rendered, the foreshortening factor $\mathbf{n}(\mathbf{x}) \cdot \mathbf{l}$ is typically included in the measured BTF data. Even if only local illumination models are used, evaluation per fragment requires a huge amount of computational power. Fortunately, todays GPUs feature such enormous processing powers that even extensive computations per fragment are possible in real-time.

Another problem related to rendering BTFs is the huge amount of storage required. In Chapter 3 a compression method based on reflectance fields (RFs) is described. For every view direction $\mathbf{v}$ from the set of sampled directions $\mathcal{M}_r$, the BTF is interpreted as a RF

$$\mathrm{RF}_{\mathbf{v}}(\mathbf{x},\mathbf{l}) = \rho_d(\mathbf{x}) + \rho_{s,\mathbf{v}}(\mathbf{x}) \sum_{j=1}^{k} \left(t_{\mathbf{v},j}(\mathbf{x}) \cdot \mathbf{l}\right)^{n_{\mathbf{v},j}(\mathbf{x})}. \tag{9.2}$$

Rendering from this compressed representation, assuming a finite number of light sources and a local illumination model, therefore requires evaluation of the following equation

$$\begin{aligned} L_o(\mathbf{x},\mathbf{v}) &= \sum_{\mathbf{l}_j \in \Lambda} \left( \sum_{v \in \mathcal{N}(\mathbf{v})} w_v(\mathbf{v})\, \mathrm{RF}_v(\mathbf{x},\mathbf{l}_j) \right) E_i^j(\mathbf{x},\mathbf{l}_j) \Big( \mathbf{n}(\mathbf{x}) \cdot \mathbf{l} \Big) \\ &= \sum_{v \in \mathcal{N}(\mathbf{v})} w_v(\mathbf{v}) \sum_{\mathbf{l}_j \in \Lambda} \mathrm{RF}_v(\mathbf{x},\mathbf{l}_j) E_i^j(\mathbf{x},\mathbf{l}_j) \Big( \mathbf{n}(\mathbf{x}) \cdot \mathbf{l}_j \Big), \end{aligned} \tag{9.3}$$

where $\Lambda$ denotes the set of directions at which the light sources contribute incoming radiance, $E_i^j$ the irradiance due to light source $j$, $\mathcal{N}(\mathbf{v})$ the index set of view directions from the measured BTF that are neighboring $\mathbf{v}$, and $w_v$ weights for interpolating values for a view direction that was possibly not measured. Please note that the current view direction $\mathbf{v}$ denotes a different entity than the index $v$ to the view directions from the measured data.

## 9.2 Point and Directional Light Sources

Typically scenes in VR systems are lit by combinations of point and directional light sources since these are directly supported by real-time graphics APIs. To render objects covered with BTFs in such environments, straightforward evaluation of Equation 9.3 has to be done per fragment by a combination of vertex and fragment shaders.

Figure 9.1 gives an overview of the rendering process. For every vertex, texture coordinates and a local coordinate system (i.e., a normal and a tangent) are specified. In the vertex shader, the current view and light directions are transformed into the local frame. The rasterizer stage then interpolates texture coordinates and directions per fragment. In the fragment shader, first the indices $v_1, v_2, v_3$ and weights $w_{v1}, w_{v2}, w_{v3}$ of the three view directions from the BTF measurement closest to the current local view direction are looked up. Next, depending on the current texture coordinates, the closest view direction, and the current MIP level, parameters of the respective reflectance fields

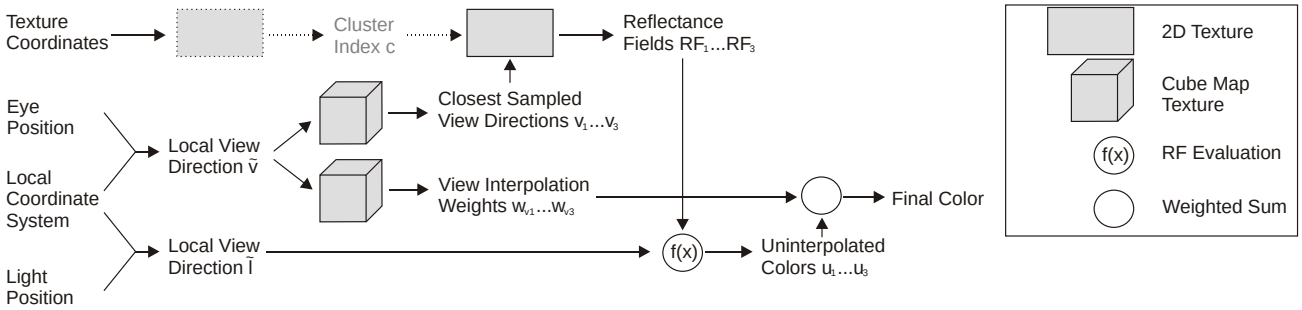



**Fig. 9.1: Data flow of rendering algorithm for scenes containing point or directional light sources.**

$\mathrm{RF}_1, \mathrm{RF}_2, \mathrm{RF}_3$ are looked up. If clustering is used, a cluster index $c$ needs to be determined for the current texture coordinate beforehand. Then, the RF parameters and the current light direction are used to compute shading values $u_1, u_2, u_3$ for the three RFs. Finally, the fragment's color is interpolated from the values $u_1, u_2, u_3$ and displayed on screen after applying suitable tone mapping [102]. If a multi-lobe approximation is employed, the parameters of more lobes have to be looked up and evaluated.

In order to lookup the weights $w_{vi}$ for the specific RFs given a view direction $\mathbf{v}$, weights and respective identifiers for the RFs are stored in one cube map each. The parameters determining the shapes of the lobes for a given reflectance field $\mathrm{RF}_v$ are stored in 2D textures. All 2D textures representing the various sampled view directions are stacked, arriving at a 3D texture. This representation permits that a single texture be bound only. In an analogous way, the view-dependent, specular albedo are stored as RGB color values in a 3D texture. Storage of the diffuse albedo requires a 2D texture only. While the lobe parameters should be stored as 32 bit floating point values, 16 bit values suffice to store HDR colors.

The memory requirements of about 190 MB for a $256^2$ BTF with 81 two-lobe RFs are very high but can be reduced significantly if clustering is employed: About 3 MB are necessary to store 1024 clusters containing 81 RFs (which corresponds to a full apparent BRDF) each. Alternatively, texture synthesis methods can be used (see below). In both cases, additional cluster index maps need to be stored, requiring 2 Bytes per sampled surface point.

In Figure 9.2 a comparison between the rendering quality achieved with a combination of a diffuse texture and a bump map, and the quality achieved with rendered BTFs represented as RFs is given. Obviously, bump maps are capable of achieving a depth impression of the material but fail to reproduce view- or light-dependent color changes. More images showing BTF covered objects are shown in Figure 9.3.

A comparison between the rendering quality achieved with various BTF rendering techniques is shown in Figure 9.4. Apparently RF rendering yields better quality than the techniques of McAllister et al. [347] and Daubert et al. [89]. Interestingly, the

**Fig. 9.2: Comparison of the quality achieved by a combination of a diffuse texture and bump-mapping (left) and BTF rendering (right). The same light configurations were used in both pictures. With BTFs the 3D structure of the corduroy material on the car seat appears realistic, while bump-mapping clearly misses the highlights for grazing light angles.**

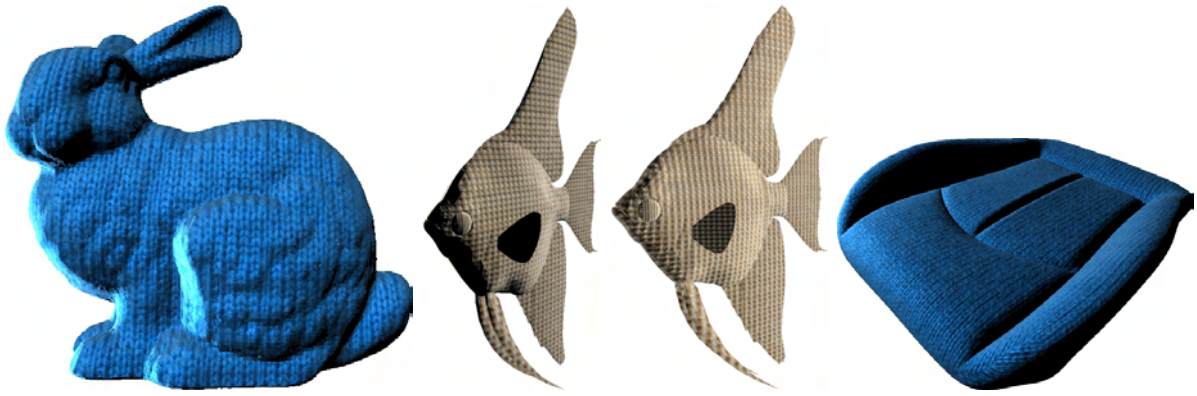


**Fig. 9.3: Several objects covered with BTFs. Left: a woolen bunny, center: a scaly fish, right: a car seat covered with knitted wool.**

quality is similar to the one achieved with per-view or per-cluster factorization, which are significantly slower to render (for more examples and an in-depth comparison see [361]). Images as the ones shown in Figure 9.2 can be computed at about 21 frames per second (fps) on an nVIDIA GeForce FX 5800 graphics card and at significantly faster frame rates on newer cards which feature much higher fill rates (e.g., about 100 fps on an nVIDIA GeForce FX 7800 at a resolution of $640 \times 480$ pixels and about 65 fps at $1280 \times 1024$ pixels). Rendering BTFs represented as RFs is therefore suitable for applications like games which need to run at real-time on a large variety of graphics boards and at various resolutions.

Unfortunately, computing outgoing radiance from the light parameters and the fitted lobe values sometimes leads to single, incorrect pixel colors (either black or white). Reasons are floating point overflows in the graphics hardware during rendering, which is

**Fig. 9.4: Comparison of the quality achieved for a shirt made of knitted wool rendered with different rendering techniques. From left to right: lit texture, Lafortune lobes [347], scaled Lafortune lobes [89], RFs, per-view factorization [447], and per-cluster factorization [369].**

partially due to the specific graphics hardware and partially to the rather high exponents for spatially restricted lobes. Using 16 bit floating point values, these problems occur more frequently.

## 9.3 Goniometric Point Light Sources

The approach for BTF rendering with point light sources can almost trivially be extended to support goniometric lights. Goniometric diagrams are sampled into cube maps, which can easily represent the full sphere of directions. During rendering, the light intensity arriving from a goniometric light source is determined by computing the inverse light direction and querying the cube map for the respective value from the goniometric diagram. This approach was first described by Dmitriev et al. [111].

Figure 9.5 shows the improvement in rendering quality when using goniometric light sources instead of point lights. If the light source is modeled as a point light source (i.e., if light intensity is emitted equally into all directions) the resulting images show a very synthetic look due to the large areas of almost equal brightness. Employing the goniometric values, the appearance becomes much more realistic since the original radiation properties are taken into account, leading to unlit regions in the rendered image. Especially important, the orientation of the light source significantly influences the final image, which is not the case using point light sources. Fortunately, the overhead for including goniometric light sources is almost negligible (one cube map per light source is required and the fragment shader needs to perform an additional texture lookup per light source). Therefore, it seems likely that such functionality be supported by future graphics APIs.

**Fig. 9.5: Rendering with point light sources emitting radiance equally into all directions (left) or according to a goniometric diagram (other). Light sources are rendered as textured cubes, their goniometric diagrams are shown as cube-map crosses.**

## 9.4 Environmental Lighting

While simple light sources yield sufficient results in many cases, they cannot represent natural lighting as, e.g., in outdoor scenes. This is far better accomplished by Image-Based Lighting techniques [93]. Since RFs are 2D functions, efficient environmental lighting is possible by a prefiltering technique similar to the one described by McAllister et al. [348]. Assuming distant lighting and neglecting view interpolation for the moment, the reflection equation with RFs changes to

$$\begin{aligned} L_o(\mathbf{x},\mathbf{v}) &= \sum_{\mathbf{l}_j\in\Lambda}\left(\rho_d(\mathbf{x})+\rho_{s,\mathbf{v}}(\mathbf{x})\sum_{j=1}^{k}\left(t_{\mathbf{v},j}(\mathbf{x})\cdot\mathbf{l}_j\right)^{n_{\mathbf{v},j}(\mathbf{x})}\right)L_i^j(\mathbf{l}_j)\Big(\mathbf{n}(\mathbf{x})\cdot\mathbf{l}_j\Big) \\ &= \rho_d(\mathbf{x})\sum_{\mathbf{l}_j\in\Lambda}L_i^j(\mathbf{l}_j)\Big(\mathbf{n}(\mathbf{x})\cdot\mathbf{l}_j\Big)+ \\ &\quad \rho_{s,\mathbf{v}}(\mathbf{x})\sum_{\mathbf{l}_j\in\Lambda}\left(\sum_{j=1}^{k}\left(t_{\mathbf{v},j}(\mathbf{x})\cdot\mathbf{l}_j\right)^{n_{\mathbf{v},j}(\mathbf{x})}\right)L_i^j(\mathbf{l}_j)\Big(\mathbf{n}(\mathbf{x})\cdot\mathbf{l}_j\Big). \end{aligned}$$

For the diffuse part of illumination, the sum of products of incoming radiance and foreshortening term can be precomputed into the prefiltered, diffuse environment map $D$. The same is possible for the specular part if the effect of the foreshortening factor is approximated [259]. Yet, the specular, prefiltered environment map

$$S\left(\frac{t_{v,j}(\mathbf{x})}{\|t_{\mathbf{v},j}(\mathbf{x})\|},n_{\mathbf{v},j}(\mathbf{x})\right)$$

still depends both on the lobe direction and the lobe exponent and therefore represents a 3D function.

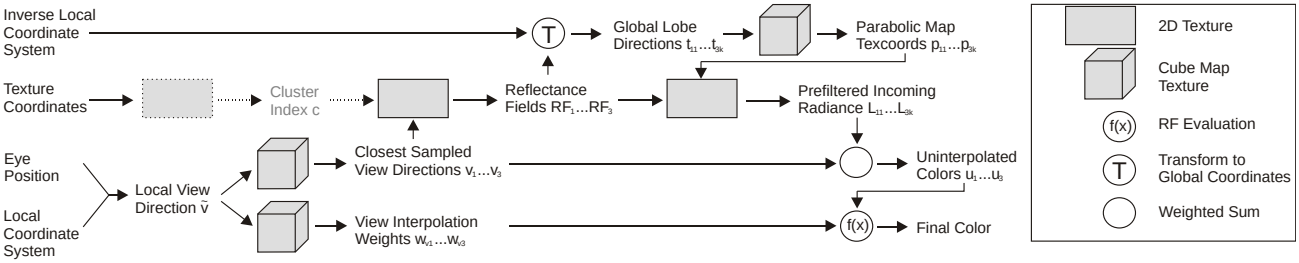



**Fig. 9.6: Data flow of rendering algorithm for scenes lit by environmental lighting.**

Using these prefiltered environment maps leads to the final rendering formula (including view interpolation)

$$L_o(\mathbf{x},\mathbf{v}) = \rho_d(\mathbf{x})D\Big(\mathbf{n}(\mathbf{x})\Big) + \sum_{v\in\mathcal{N}(\mathbf{v})}\left(w_v(\mathbf{v})\rho_{s,\mathbf{v}}(\mathbf{x})\cdot \sum_{j=1}^{k} S\left(\frac{t_{v,j}(\mathbf{x})}{\|t_{v,j}(\mathbf{x})\|}, n_{v,j}(\mathbf{x})\right)\|t_{v,j}(\mathbf{x})\|^{n_{v,j}(\mathbf{x})}\Big(\mathbf{n}(\mathbf{x})\cdot t_{v,j}(\mathbf{x})\Big)\right).$$

Please note that in this case $t_{v,j}$ is specified w.r.t. the global coordinate system since rotation of the lighting environment to the surface point's local coordinate system is inefficient.

The prefiltered environment maps are made available for rendering as textures. While the prefiltered, diffuse map should be loaded as a cube map, the chosen texture format for the specular map depends on the possibilities of the graphics board. To encode HDR environments, the texture target needs to support floating point values, optimally additionally bi- or even trilinear interpolation. Currently, reasonable choices are MIP-mapped cube maps (one MIP level for each of the sampled exponents $n = 2^{\frac{i}{2}}$ for $i = 0\ldots10$) or a 3D texture with layers corresponding to dual-parabolic maps [216], one layer per sampled exponent.

The rendering process is depicted in Figure 9.6. The inputs are the same as the ones used for handling point light sources, but additionally a transformation matrix rotating the local coordinate system to the global frame is provided. As previously, this data is specified per vertex. Per-fragment interpolation is done by the rasterizer.

Computing the local view direction, determining closest measured view directions and interpolation weights, and fetching the RF parameters is analogous to the rendering algorithm for simple light sources. To evaluate image-based lighting, the lobe directions $t_{v,j}$ are rotated into the global coordinate system (in which the lighting environment is defined as well) first. The resulting directions $\mathbf{t}_{ij}$ are used to first determine parabolic map texture coordinates $\mathbf{p}$ which – combined with the lobe exponent – serve as indices

**Fig. 9.7: Example of Image-Based Lighting. Right: A shirt covered with an aluminium material in the Galileo environment. Left and center: diffuse and specular part of appearance.**

to lookup the incoming radiances $L_{ij}$ from the prefiltered environment maps. Finally, the incoming radiances for the various lobes are scaled according to the exponentiated lengths of the respective lobes, weighted by the approximated foreshortening term, and summed. View interpolation is again the same as for the case of simple light sources.

Figure 9.7 shows some rendered images of a shirt covered with the shiny, aluminium BTF lit by the Galileo environments. Such images render at about 14 fps on an nVIDIA GeForce FX 5800 at a resolution of 640 $\times$ 480 pixels. On newer graphics boards like the nVIDIA GeForce FX 7800, performance is boosted to about 60 fps for resolutions of 1280 $\times$ 1024 pixels, which is absolutely sufficient for real-time applications.

Derivation of the rendering equation for BTF textured objects lit by distant lighting environments contains an approximation of the area-foreshortening term [259]. The accuracy of this approximation increases with the specularity of the BTF. Therefore, best results are achieved with the presented technique if very shiny materials are handled, like the aluminium BTF. Although limiting applicability of the method, this still enables handling of a large number of materials: Almost diffuse materials are accurately represented by the diffuse prefiltered environment map, specular materials by the specular counterpart.

Memory requirements for this rendering technique are comparable to the one for image-based lighting. Only the prefiltered environment maps have to be stored in addition, which requires an additional 720 kB if 128$\times$128 pixel dual-parabolic maps are used. As in the case of point-based lighting, rendered images may show shading artifacts due to numerically instable evaluations of non-linear lobes.


**Fig. 9.8: Data for rendering tiled BTFs. Ten tiles (left) arranged in a single texture $T_p$ (center) with $(c_{T_p}, r_{T_p}) = (5, 2)$, right: tile matrix $T_i$ with $(c_{T_i}, r_{T_i}) = (9, 6)$ storing offsets for respective tiles in $T_p$ (tile numbers are printed in large, gray numbers).**

## 9.5 Synthesized BTFs

The original method for rendering BTFs represented by RFs [358] contained no support for clustered apparent BRDFs but instead employed texture synthesis techniques to reduce storage requirements. Starting from BTF samples with rather small spatial extent (e.g., 64×64 or 96×96 pixels) arbitrary large textures were synthesized. Instead of storing per-pixel ARBDFs in the synthesized BTF, indices into the original, smaller BTF are stored.

The approaches based on clustering and texture synthesis are equivalent insofar that they use a set of predefined ABRDFs. While it is chosen optimally with the clustering method, it is defined implicitly if texture synthesis is used. Therefore, rendering synthesized BTFs is supported in the same way as clustered BTFs: Before accessing the RF parameters, the index to the ABRDF corresponding to the current texture coordinate has to be looked up. Yet, due to the more efficient choice of base ABRDFs when using clustering, methods based on texture synthesis from small patches should not be used. Instead, textures should be synthesized from large patches and the results should then be clustered.

## 9.6 Tiled BTFs

While a combination of standard texture synthesis methods and clustering yields good results for patches of medium size, memory requirements become too big if large areas need to be textured. In such a case, tile based synthesis is the best choice at hand because it enables hierarchical storage of required information (cf. Chapter 4): Texture values of tiles are coded explicitly, but coding the arrangement of tiles requires storage of tile indices only.

To handle such tiled representations during rendering, multiple tiles need to be available and the mapping from texture coordinates to individual tiles needs to be specified. The first requirement is easily fulfilled by packing multiple tiles into one texture $T_p$ (cf. left and center part of Figure 9.8). If clustering is employed, $T_p$ contains cluster indices instead of color values. If the tiles are of identical size, the mapping from texture coordinates $(s,t)$ to a tile index and a texture coordinate within the tile (and thus a texture coordinate $(s_p,t_p)$ w.r.t. $T_p$) can be computed as

$$\begin{aligned}(s',t') &= \mathrm{frac}\left(\frac{(s,t)}{(c_{T_i},r_{T_i})\cdot(w_t,h_t)}\right)\cdot(c_{T_i},r_{T_i})\\ (s_p,t_p) &= T_i\left(\frac{\lfloor(s',t')\rfloor}{(c_{T_i},r_{T_i})}\right)+\frac{\mathrm{frac}\Big((s',t')\Big)}{(c_{T_p},r_{T_p})}\end{aligned}$$

where $(w_t,h_t)$ denote the width and height of a single tile, $(c_{T_i},r_{T_i})$ and $(c_{T_p},r_{T_p})$ the number of columns and rows of tile matrix $T_i$ and the texture $T_p$ containing all tiles (cf. right and center part of Figure 9.8), $\mathrm{frac}$ is a function computing the fractional part of a number, and operations are defined component-wise. $(s',t')$ can be thought of as texture coordinates for the tile matrix, with the integer parts identifying tile offsets from $T_i$, and the fractional parts referencing texels within the tile. Application of the $\mathrm{frac}$ function for computing $(s',t')$ implements texture repetition.

In the case when tiles are created such that they can be placed next to each other randomly, MIP mapping of the tiles can be achieved by simply MIP mapping $T_p$. This solution was also applied by Wei [571], who developed a similar approach based on Wang Tile textures [73] in concurrent work.

One of the big advantages of tile-based texture synthesis is that memory requirements increase just very slightly since hierarchical ways for specifying textures can be employed. To render from 1024 clusters of ABRDFs and one 256×256 tile, 3.1 MB are required. With the same number of clusters, ten 256×256 tiles, and a tile matrix of 16×16 the impression of a 4096×4096 texture is achieved. This requires about 4.3 MB, which is much less than the 44 MB required if 4096×4096 indices are stored explicitly. Fortunately, the run-time overhead when rendering from tiled textures is negligible.

Figure 9.9 shows that tiled BTFs clearly reduce the amount of visible repetition, e.g., compared to the images in Figure 9.4, leading to significantly improved rendering quality. Unfortunately, the size of tiles limits the length or area of visible structures like the leather's grooves.

**Fig. 9.9: Car seat covered with two types of leather. Tile-based synthesis clearly reduces repetition artifacts.**

## 9.7 Discussion

In this chapter methods for real-time rendering from compressed BTF materials were presented. Combinations with directional or point light sources, with goniometric lights, and lighting environments are possible without impacting the performance of the method. Additionally, integration with texture synthesis approaches was demonstrated which allows for real-time rendering of textured surfaces without notable texture repetitions. Despite the development of novel, more accurate BTF compression techniques, rendering from RFs is still highly compatible since it offers a very good tradeoff between rendering speed and BTF reconstruction quality.

Obviously the methods demonstrated in this chapter do not suffice to achieve real-time predictive rendering, especially for complex lighting environments including global lighting effects. Therefore, methods adding such global effects are described in the following two chapters.

# Chapter 10

# Real-Time Rendering from Precomputed Data

Real-time rendering algorithms for compressed BTF representations based on local lighting models as the one presented in the previous chapter are extremely important since they can easily be used in existing VR systems which are usually based on local illumination models. One of the key aspects justifying this widespread use is negligible precomputation time. Yet, obviously such approaches are not capable of achieving predictive rendering due to omission of global lighting effects.

In the state of the art overview of this part a large number of methods rendering global illumination solutions at interactive frame rates was presented. For the application area specifically targeted in this thesis, virtual prototyping, combinations of methods rendering precomputed global illumination solutions and methods simulating high-frequency effects due to specular materials were determined most suitable. Therefore, such techniques are described in the following. While the next chapter focuses on interactive handling of specular materials, this one concentrates on rendering from precomputed data by describing novel methods based on Surface Light Field data or Precomputed Radiance Transfer.

# 10.1 SLF Rendering

Surface Light Fields (SLFs) describe the appearance of surfaces as a 4D function $\mathrm{SLF}$ storing the outgoing radiance at surface position $\mathbf{x}$ into local direction $\mathbf{v}$. The function can be derived from the Rendering Equation by fixing geometry, surface materials, and lighting:

$$\mathrm{SLF}(\mathbf{x}, \mathbf{v}) = L_e(\mathbf{x}, \mathbf{v}) + \int_{S^2} \rho(\mathbf{x}, \mathbf{v}, \mathbf{l}) L_i(\mathbf{x}, \mathbf{l}) \Big( \mathbf{n}(\mathbf{x}) \cdot \mathbf{l} \Big) d\mathbf{l}.$$

In the same way BTFs can either be considered textures containing per-texel apparent BRDFs or sets of textures for corresponding view and light directions, SLFs are either handled as textures with per-texel *lumispheres* or as view-dependent textures.

SLFs have been employed to handle both purely virtual [363, 62] and captured real [586, 62] data, with more recent focus on real data (i.e., 3D photography), since it can be captured efficiently by taking a number of photographs of real objects in controlled environments. As previous results showed, rendered images represent very accurate replications of reality. Thus, in the context of this thesis, SLF rendering represents a suitable way to produce very precise and possibly even predictive results.

In combination with synthetic data, two main problems hinder widespread use of SLF rendering. First, as shown above, precomputation of SLF data requires computing solutions to the Rendering Equation of a scene. While such solutions require extensive computations when computing a single view of the scene already, even more effort has to be spent on construction of SLFs. This makes SLF data unattractive for application areas that require fast changes to either the geometry, the materials, or the lighting (e.g., virtual design).

The second problem arises from the large amount of data contained in a SLF, commonly several GBs even for relatively simple scenes and moderate sampling rates for surface position and view direction. Fortunately, typical data sets can be compressed using standard compression techniques without reducing the rendering quality too much.

In the following a novel approach targeting these problems is presented. It reduces precomputation time by deriving incoming light fields from the global illumination solver which can be convolved with surface materials quickly, enabling much more efficient material replacement. Additionally, SLF data is compressed using a more efficient technique than previous approaches, allowing for efficient real-time rendering using graphics hardware.

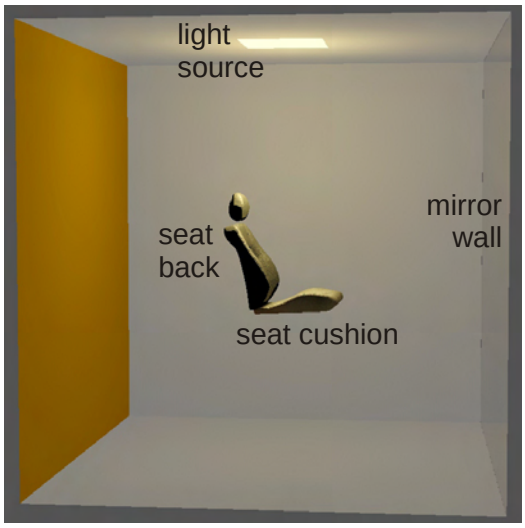



**Fig. 10.1: Test scene for SLF rendering. A car seat is placed in an open box consisting of a reddish wall, a mirror wall, three diffuse white walls, and a yellowish area light source.**

### 10.1.1 Data Preparation

SLF data for virtual objects is commonly derived by computing solutions to the rendering equation. While being a computationally expensive operation, it additionally requires the geometry, materials, and light to be fixed in advance. To relax this problem for application areas that require fast exchange of surface materials (i.e., interior design), data generation can be shortened by computing incoming surface light fields (i.e., SLFs storing incident instead of exitant radiance). Having such data at hand, outgoing SLFs can be computed by combining incident SLFs and surface material. This is much faster since expensive computations including global effects need to be performed only once, while computation of outgoing SLFs requires local operations only.

An additional speedup can be achieved by two methods. The first is based on transforming incoming SLFs and materials into appropriate bases. Following ideas from Precomputed Radiance Transfer rendering, encoding incident lumispheres and materials by truncated series of Spherical Harmonics (SH) or Haar wavelets allows for linear- or non-linear approximations with negligible errors and greatly reduced SLF construction time. As an example, generating an outgoing SLF of a spatial resolution of $1024^2$ texels and a hemicube of 768 directions for incoming and outgoing SLFs requires evaluating the integral part of

$$\mathrm{SLF}_{out}(\mathbf{x},\mathbf{v}) = L_e(\mathbf{x},\mathbf{v}) + \int_{S^2} \rho(\mathbf{x},\mathbf{v},\mathbf{l})\,\mathrm{SLF}_{in}(\mathbf{x},\mathbf{l})d\mathbf{l}.$$

about 620 billion times, requiring more than $1.2$ trillion arithmetic operations to be executed per color channel. Projecting onto an SH or Haar Wavelet basis and retaining

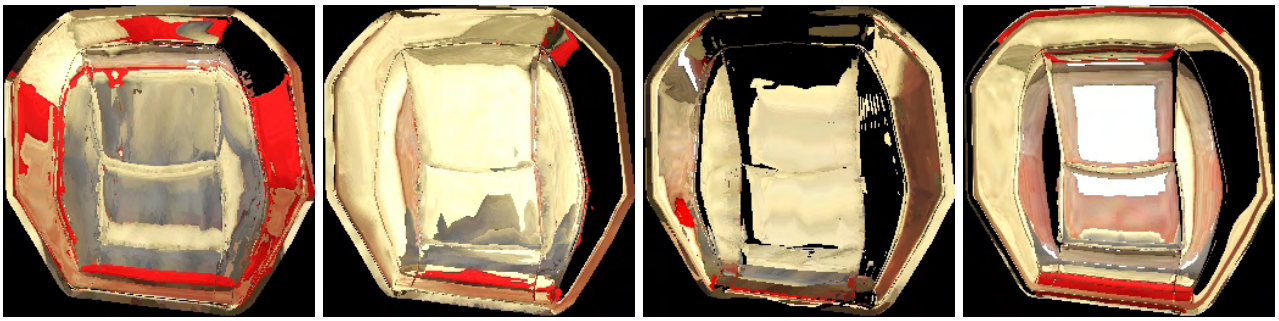


**Fig. 10.2: Some slices of the seat cushion's incoming SLF. Colors encode the amount of light reaching the surface from a specific direction (one fixed local direction for each image).**

25-100 coefficients leads to a speedup by a factor of about 50-800.

Figure 10.1 shows an example scene consisting of a car seat placed in an open box with an area light on top, a red wall on the left and a mirror wall on the right. A high-resolution SLF was computed for the car seat cushion (cf. Figure 10.2), which required about 200 hours using a highly accurate yet slow raytracing engine and a single CPU. Computing an outgoing SLF from the incoming SLF requires about 80 hours if SLFs are stored with directional bases. Projecting onto 6th order SH reduces the computation time to about 25 minutes. Retaining about 40 Haar basis coefficients allows for similar reductions in precomputation time.

A second, different method to speed up the process follows ideas for efficient rendering of objects covered with compressed BTF materials in lighting environments [370]. Both $\mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l})$ and incident lighting $\mathrm{SLF}_{in}(\mathbf{x},\mathbf{v})$ are compressed using clustered PCA, yielding

$$\begin{aligned}\mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) &\approx \sum_c w_{m_c}\Big(k_1(\mathbf{x})\Big)\rho_{c,k_1(\mathbf{x})}(\mathbf{v},\mathbf{l}),\\ \mathrm{SLF}_{in} &\approx \sum_d w_{l_d}\Big(k_2(\mathbf{x})\Big)\lambda_{d,k_2(\mathbf{x})}(\mathbf{l}),\end{aligned}$$

where $k_1$ and $k_2$ select the cluster for the given surface point, $w_m$ and $w_l$ denote material and lighting weights, $\rho$ Eigen-ABRDFs, and $\lambda$ Eigen-Lumispheres. Having these compressed representations, the above integral changes to

$$
\begin{aligned}
\mathrm{SLF}_{out}(\mathbf{x},\mathbf{v}) &= L_e(\mathbf{x},\mathbf{v}) + \\
&\int_{S^2} \left( \sum_c w_{m_c}\Big(k_1(\mathbf{x})\Big) \rho_{c,k_1(\mathbf{x})}(\mathbf{v},\mathbf{l}) \right) \left( \sum_d w_{l_d}\Big(k_2(\mathbf{x})\Big) \lambda_{d,k_2(\mathbf{x})}(\mathbf{l}) \right) d\mathbf{l} \\
&= L_e(\mathbf{x},\mathbf{v}) + \sum_c \sum_d w_{m_c}\Big(k_1(\mathbf{x})\Big) w_{l_d}\Big(k_2(\mathbf{x})\Big) \int_{S^2} \rho_{c,k_1(\mathbf{x})}(\mathbf{v},\mathbf{l}) \lambda_{d,k_2(\mathbf{x})}(\mathbf{l}) d\mathbf{l} \\
&= L_e(\mathbf{x},\mathbf{v}) + \sum_c \sum_d w_{m_c}\Big(k_1(\mathbf{x})\Big) w_{l_d}\Big(k_2(\mathbf{x})\Big) P_{c,d,k_1(\mathbf{x}),k_2(\mathbf{x})}(\mathbf{v}),
\end{aligned}
$$

with $P$ being the precomputed interaction between light and material clusters. Efficiency is gained since evaluation costs per surface point reduce to about $3 \cdot C \cdot D$ operations, with $C$ and $D$ being the number of components from local PCA. Thus, utilizing a high-quality setting of 4 BTF components and 8 lighting components, about 100 operations need to be performed for each sampled surface point. The additional costs for compressing BTFs and incident lumispheres are relatively low (about 2.5 hours for a BTF of $256 \times 256$ texels and about 1 hour for the lumispheres) and can partially be avoided if BTFs are provided in this compressed format already.

Both approaches have their specific advantages and disadvantages. Both allow for significant speedups by sacrificing correct interreflections (i.e., interreflections do not contain the properties of exchanged materials) which makes them suitable for relatively diffuse objects only. Nevertheless, often such interreflection effects can safely be ignored. While the first approach is faster w.r.t. total computation time, the second one does not require explicit storage of the outgoing SLF. Evaluating the double sum can directly be implemented on graphics hardware as a fragment shader. Since the first approach enables more efficient rendering from the precomputed data on simpler graphics hardware, this techniques is chosen in the following.

### 10.1.2 Data Compression

After computing outgoing SLFs (cf. Figure 10.3 for SLF slices from the example scene) interactive rendering is possible in principle. Yet, the large amounts of data (e.g., about 220 MBs are required to store the 81 slices for the seat cushion as half precision, losslessly compressed OpenEXR [231] images) typically require significant data compression to make interactive handling possible.

Existing approaches for compression of SLF data were based on vector quantization or PCA, or PCA followed by vector quantization. While these approaches achieve relatively good result, results from BTF compression showed that compression based on local PCA [371] achieve significantly better results than aforementioned techniques.

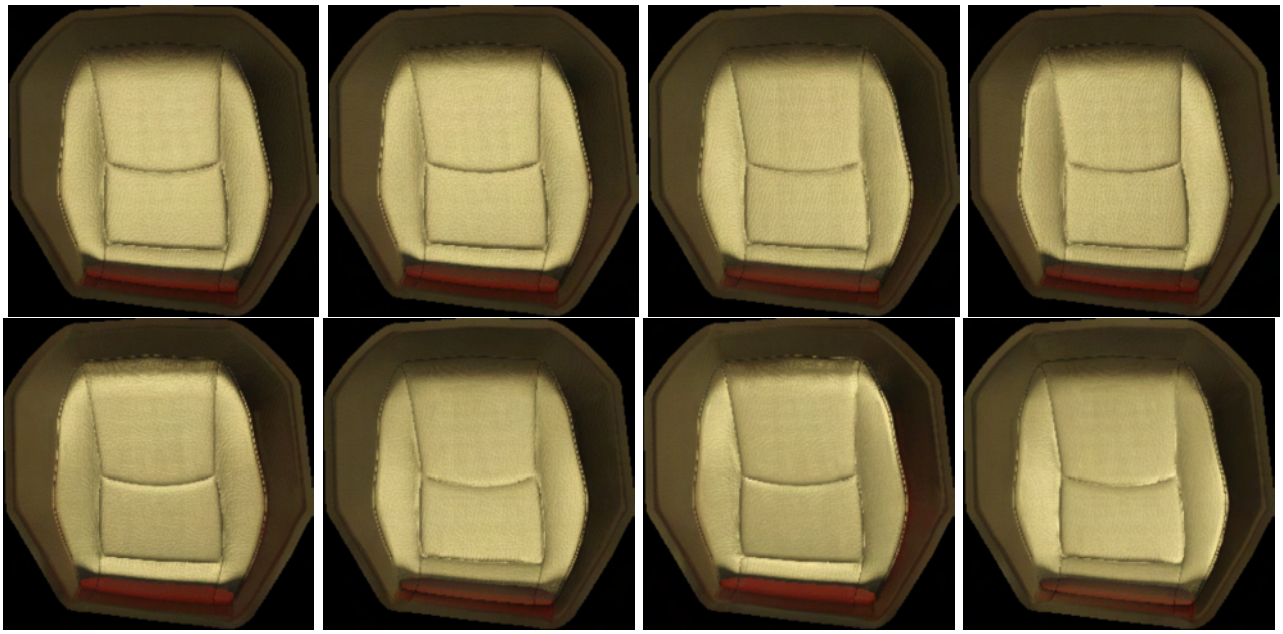


**Fig. 10.3: Some slices of the seat cushion's outgoing SLF. The seat is covered with a light artificial leather.**

| | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| 2 | 13.7% | 12.5% | 11.0% | 10.8% | 10.4% |
| 4 | 8.1% | 7.3% | 6.5% | 6.4% | 6.4% |
| 8 | 4.6% | 3.9% | 3.5% | 3.3% | 3.2% |

**Tab. 10.1: RMS reconstruction error of compressed SLFs for varying numbers of clusters (columns) and components (rows).**

Therefore, this approach applies local PCA to compress precomputed outgoing SLFs. Using a high-quality setting of 128 clusters and 4 components, the storage requirements for the SLFs of the car seat can be reduced to about 9 MBs. The storage requirements scale approximately linearly with the number of components and are almost independent of the number of clusters. Table 10.1 gives an overview of the RMS reconstruction errors for various configurations of clusters and components. During rendering, errors of below 7% are hardly visible. Compared to the numbers for local PCA compression of BTF materials [369] these numbers are relatively high since the data includes additional high-dynamic range light variation, decreasing correlation between contained elements. Figure 10.4 shows a visual comparison between rendering from compressed and uncompressed data. Although errors are visible at close inspection, they are hardly recognizable from typical distances.

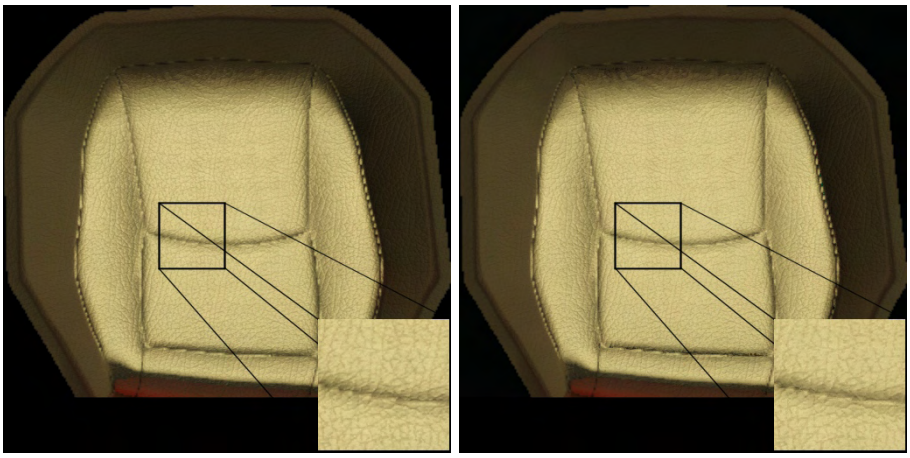


**Fig. 10.4: Reconstruction quality from local PCA compressed representation. Left: original, right: compressed.**

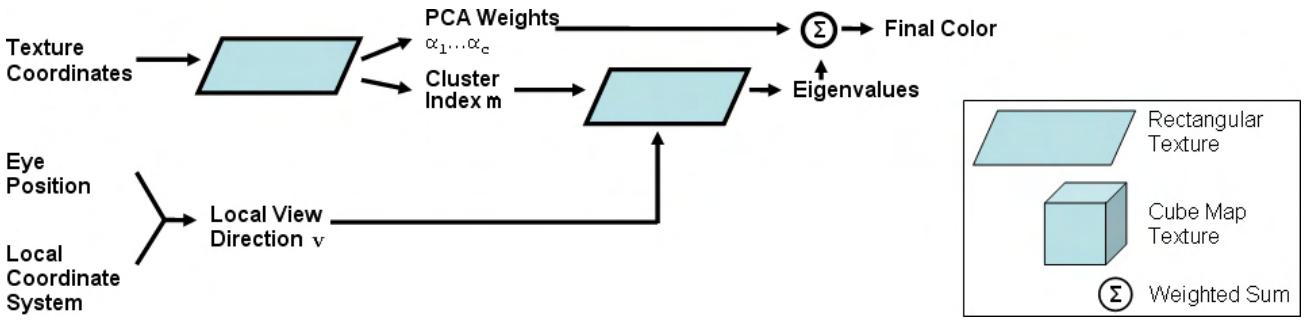



**Fig. 10.5: Scheme for implementation of SLF rendering on programmable GPUs.**

## 10.1.3 Rendering

Rendering SLFs is a very simple task since it requires lookup and interpolation of exitant radiance values corresponding to surface position and view only. Figure 10.5 gives an overview of the scheme implementing rendering per fragment, including decoding of compressed SLF values. For every vertex, texture coordinates and a local coordinate system (i.e., a normal and a tangent) are specified. In the vertex shader, the current view direction is transformed into the local frame. The rasterizer stage then interpolates texture coordinates and directions per fragment. In the fragment shader, first the lumisphere cluster $m$ corresponding to the current texture coordinate is looked up, together with the corresponding PCA weights $\alpha_i$. Next the exitant radiance values corresponding to the current view direction and luminance cluster are determined, one RGB triple $r_{m,i}$ for each eigen-lumisphere. Finally, the actual exitant radiance values are reconstructed from the weights $\alpha_i$ and the triples $r_{m,i}$.

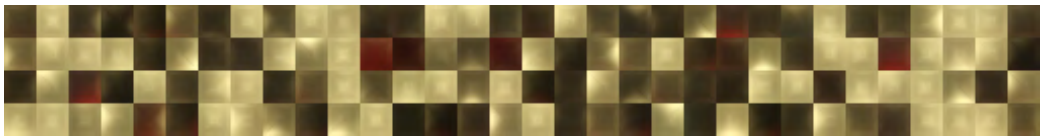


**Fig. 10.6: Packing mean lumispheres into a single texture.**

Since exitant radiance values need to be read for arbitrary view directions and since SLFs are sampled for a limited number of directions only, interpolation is required. Since decompression of eigen-lumispheres is a linear operation, linear interpolation of exitant radiance values prior to reconstruction will achieve the same results as linear interpolation of reconstructed values. This interpolation can efficiently be performed by graphics hardware if the data is sampled regularly. Therefore, the eigen-lumispheres are resampled at a set of directions corresponding to an $16 \times 16$ parabolic map [216] and are packed into a single texture (cf. Figure 10.6).

Figure 10.7 shows rendered images of the test scene, tone mapped to produce displayable images. For comparison, the seat's back is rendered with a direct lighting model, lit by a point light positioned in the center of the area light source. It is clearly visible that SLF rendering offers all global illumination effects: shadows (e.g., bottom left image: shadow of seat's back onto cushion), indirect lighting (the parts of the cushion oriented downwards are lit while downwards pointing parts of the seat's back are dark), and color bleeding (see top row, second image from left: the back looks red like the wall behind it). All of these effects are rendered at texture resolution accuracy, which is a big advantage over standard PRT rendering methods, which interpolate illumination values between mesh vertices. Images as the ones shown can be rendered at 40 frames per second on a standard GeForce FX 7800 graphics board at full-screen resolution ($1280 \times 1024$ pixels).

### 10.1.4 Discussion

The use of SLF data adds significant realism to rendered scenes by reproducing global illumination effects. These effects include all-frequency lighting and complex glossy-diffuse materials, which are both reproduced at texture resolution, allowing for coarse object triangulations. As shown, rendering SLF data can efficiently be done in real-time on graphics hardware. Storage requirements for relatively small scenes are relatively low. Thus, SLF rendering is highly suitable for interactive inspection of global illumination solutions of small to mid-sized objects.

Unfortunately, SLF rendering has several drawbacks as well. First, precomputation requires a large amount of time. The proposed approach for reducing precomputation

**Fig. 10.7: SLF rendering of the test scene. For comparison, lighting of the seat's back was computed with a local lighting model (point light source centered in area light).**

time by reusing precomputed illumination reduced this problem to some extent. Another improvement can be achieved when running the global illumination solver on a cluster of CPUs, which is easily possible with most available renderers. Combining these two approaches, SLFs can be computed in few hours.

A second problem is related to the large amount of SLF data. As shown above, existing compression techniques reduce storage requirements to moderate amounts for relatively small objects. Unfortunately, the amount scales linearly with texture area, which increases significantly when handling larger scenes or requiring higher resolutions. To handle such cases, even better compression techniques need to be devised in the future. Possible approaches may either decrease the per-texel storage requirements by improving data correlation as proposed by Müller et al. [372], or should optimally decouple storage requirements from texture resolution.

A third, inherent limitation of SLF rendering is that the scene cannot be changed. If users require that changes to the scene should be possible, other techniques are more suitable. One such technique based on all-frequency PRT is presented in the following section.

## 10.2 PRT Rendering

Rendering methods based on Precomputed Radiance Transfer are typically employed to simulate the change of appearance of an object due to an interactively modified, distant lighting environment. Compared to methods handling SLFs, they are therefore more flexible. For many virtual prototyping applications such flexibility is greatly desired since it enables, e.g., simulations of driving or walking scenarios.

In Chapter 8 a large number of PRT rendering methods have been described, most of them focusing on very different aspects of global illumination solutions. Especially interesting in the context of this thesis are approaches supporting complex, accurate, spatially varying material representations (i.e., BTFs). So far, only two such methods exist [488, 370]. Unfortunately, both suffer from several restrictions. The approach of Sloan et al. [488] is limited to BTFs of relatively small spatial extent. Both techniques support low-frequency environmental lighting only, which prohibits sharp lighting effects. Obviously, these restrictions need to be removed for a method suitable for interactive virtual prototyping. An approach targeting this goal is described in the following.

### 10.2.1 Basic Idea

Like rendering methods based on SLF data, PRT methods precompute radiance transfer by solving the Rendering Equation for a number of sampled view directions and surface locations. The significant difference between SLF and PRT data is that the latter one supports exchangeable lighting situations while typically restricting itself to distant lighting environments. These assumptions, combined with BTF materials, lead to the following equation to be solved:

$$L_r(\mathbf{x},\mathbf{v}) = \int_\Omega BTF(\mathbf{x},\mathbf{v},\mathbf{l})V(\mathbf{x},\mathbf{l})L_i(\mathbf{l})\Big(\mathbf{n}(\mathbf{x})\cdot\mathbf{l}\Big)d\mathbf{l},$$

where $\Omega$ denotes the hemisphere of local directions, and $V$ the visibility function. While the BTF is defined for all surface locations, $V$ is typically evaluated at a coarser set of points $\{\mathbf{t}_i\}$. Usually, these coincide with the vertices of a mesh. Visibility values for arbitrary surface points $\mathbf{x}$ are then interpolated from the three closest sample points $\mathbf{t}_1$, $\mathbf{t}_2$, and $\mathbf{t}_3$. The respective interpolation weights $w_1$, $w_2$, and $w_3$ correspond to the barycentric coordinates of $\mathbf{x}$ w.r.t. the triangle spanned by $\mathbf{t}_1$, $\mathbf{t}_2$, and $\mathbf{t}_3$. Additionally assuming a finite set of distant light directions (e.g., the pixels of a cube map) leads to

the modified equation:

$$L_r(\mathbf{x},\mathbf{v}) = \sum_{\mathbf{l}_j} \mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}_j) \left( \sum_{d=1}^{3} w_d V(\mathbf{t}_d,\mathbf{l}_j) \Big( \mathbf{n}\left(\mathbf{t}_d\right) \cdot \mathbf{l}_j \Big) \right) L_i(\mathbf{l}_j).$$

Please note that light directions are not limited to the hemisphere of local directions, thus it needs to be defined that

$$\forall \mathbf{l}_j \notin \Omega . \, \mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}_j) = 0.$$

This equation can be precomputed for a suitable light basis and a sufficiently dense set of view directions and surface positions. Unfortunately, this requires sampling of a six-dimensional function which implies a huge amount of precomputation time and storage. Therefore, following ideas from BTF compression [447], the BTF is factorized into spatially dependent functions $g$ and spatially independent ones $h$:

$$\mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \sum_{c_B=0}^{C_B} g_{c_B}\left(\mathbf{x}\right) h_{c_B}\left(\mathbf{v},\mathbf{l}\right).$$

The factorized representation allows to move the spatially dependent terms in front of the sum over the directionally dependent parts. The product of the functions $h$, the visibility function, and the area-foreshortening term then becomes the transfer function $T$ which needs to be precomputed:

$$L_r(\mathbf{x},\mathbf{v}) \approx \sum_{c_B=0}^{C_B} g_{c_B}\left(\mathbf{x}\right) \sum_{d=1}^{3} w_d \sum_{\mathbf{l}_j} \underbrace{h_{c_B}\left(\mathbf{v},\mathbf{l}_j\right) V(\mathbf{t}_d,\mathbf{l}_j) \Big( \mathbf{n}\left(\mathbf{t}_d\right) \cdot \mathbf{l}_j \Big)}_{T_{c_B}(\mathbf{t}_d,\mathbf{v},\mathbf{l}_j)} L_i(\mathbf{l}_j). \quad (10.1)$$

The above equation has a significant practical disadvantage: Each time the lighting $L$ is changed, $T$ has to be recomputed. Obviously, this prohibits interactive changes of the lighting environment – one of the key ideas behind PRT rendering methods. To solve this problem, the transfer matrices $T$ are factorized into light-dependent and light-independent functions $q$ and $p$:

$$T_{c_B}\left(\mathbf{t}_d,\mathbf{v},\mathbf{l}\right) \approx \sum_{c_T=0}^{C_T} p_{c_B,c_T}\left(\mathbf{t}_d,\mathbf{v}\right) q_{c_B,c_T}\left(\mathbf{l}\right)$$

Factorization enables to move the functions $p$ in front of the sum over the light directions, leading to the final equation:

$$L_r(\mathbf{x},\mathbf{v}) \approx \sum_{c_B=0}^{C_B} g_{c_B}\left(\mathbf{x}\right) \sum_{d=1}^{3} w_d \sum_{c_T=0}^{C_T} p_{c_B,c_T}\left(\mathbf{t}_d,\mathbf{v}\right) \sum_{\mathbf{l}_j} q_{c_B,c_T}\left(\mathbf{l}_j\right) L_i(\mathbf{l}_j) \quad (10.2)$$

It has the significant advantage that only the light dependent part of the transfer function $T$ needs to be recomputed if the lighting environment is changed. This computation corresponds to an inner product of a vector representing light coefficients and a vector representing $q$. Encoding both in a suitable basis, i.e., a Haar wavelet basis, and applying non-linear optimization as in [382] leads to a small number of vector dimensions. This in turn leads to efficient evaluation of the sum, allowing the interactive use of detailed, dynamic lighting environments.

In the following, the three most important preprocessing steps (BTF and transfer function factorization, and non-linear approximation of lighting and transfer) are detailed. Then, interactive rendering schemes for the novel PRT representation are outlined and the results achieved with this method are described, identifying strengths and weaknesses. Finally, ideas for future research of PRT methods related to this approach are described.

### 10.2.2 Factorization of BTF and Transfer Function

As described in Chapter 2, several approaches for factorization of BTF data exist, which can also be applied to transfer function data. All of them feature some advantages and disadvantages. In the current setting, it is especially desirable to achieve fast reconstruction while retaining as much accuracy as possible. Therefore, factorization approaches handling subsets of the data (i.e., per-view factorization [447] or clustered factorization [369]) are suitable candidates.

Since clustered factorization approaches yield better trade-offs between reconstruction quality and data compression they appear to be a natural choice for factorizing both BTF and transfer functions. Following the approach described by Müller et al. [369], this leads to the following BTF and transfer function representations:

$$\begin{aligned} \mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) &\approx \sum_{c_B=0}^{C_B} \tilde{g}_{c_B}(\mathbf{x})\tilde{h}_{c_B,k_B(\mathbf{x})}(\mathbf{v},\mathbf{l}), \\ T_{c_B,k_B(\mathbf{x})}\left(\mathbf{t}_d,\mathbf{v},\mathbf{l}\right) &\approx \sum_{c_T=0}^{C_T} p_{c_B,c_T,k_B(\mathbf{x}),k_T(\mathbf{l},c_B,k_B(\mathbf{x}))}\left(\mathbf{t}_d,\mathbf{v}\right) q_{c_B,c_T,k_B(\mathbf{x})}\left(\mathbf{l}\right), \end{aligned}$$

where indices $c_B = 0$ and $c_T = 0$ refer to mean values of the corresponding clusters $k_B$ and $k_T$.

Unfortunately, this representation leads to a practical problem: In order to factorize the transfer functions, they need to be computed first. For typical values of about 20000 vertices, a cube map resolution of $6 \times 32 \times 32$ for encoding environmental lighting,

and $|M_r| = 81$, such a transfer function contains about 120 GB. Storing all this data to harddisk and reloading it on demand unfortunately requires such a large amount of time that this approach becomes impracticable.

Therefore, it turns out more convenient to employ per-view factorization for the BTF [447], leading to the following BTF and transfer function representations:

$$\begin{aligned} \mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) &\approx \sum_{c_B=0}^{C_B} g_{\mathbf{v},c_B}(\mathbf{x})\, h_{\mathbf{v},c_B}(\mathbf{l}), \\ T_{\mathbf{v},c_B}(\mathbf{t}_d,\mathbf{l}) &\approx \sum_{c_T=0}^{C_T} p_{\mathbf{v},c_B,c_T}(\mathbf{t}_d)\, q_{\mathbf{v},c_B,c_T,k_T(\mathbf{t}_d,\mathbf{v},c_B)}(\mathbf{l}). \end{aligned} \tag{10.3}$$

Per-view factorization yields transfer functions independent of the view direction (i.e., the functions themselves depend on light direction only but for each view direction a separate set of transfer functions is generated). Therefore, for each factorization step only about 1.5 GB of data need to be computed for the same typical settings as above. Fortunately, this amount of data can be handled in-core on modern PCs.

The per-view BTF factorization processes one matrix for each view direction $\mathbf{v}_i$, where the $j$-th column holds the $BTF(\mathbf{x},\mathbf{v},\mathbf{l}_j)$ values. Data values are centered prior to storing them in the matrix by subtracting mean BTF values

$$m_{\mathbf{v}}(\mathbf{x}) = \frac{1}{|M_r|}\sum_{j=1}^{|M_r|}\mathrm{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}_j)$$

Factorizing these matrices and retaining the $C_B$ most significant principal components each leads to the BTF representation from Equation 10.3 with $g_{\mathbf{v},0} = m_{\mathbf{v}}$ and thus $h_{\mathbf{v},0} = 1$.

Clustered factorization of the transfer functions proceedings in a similar way. From Equation 10.1 follows that a separate matrix has to be factorized for each sampled view direction $\mathbf{v}$ and each component $c_B$ of the BTF factorization. The matrices are composed of rows and columns corresponding to vertices and light directions. Clusters are determined in an initial step that applies k-means clustering [325, 369]. If $C_T$ is chosen correctly, this approach succeeds in capturing local behavior in the data (cf. Figure 10.8). As in the case of BTFs, the transfer functions are centered first by computing and subtracting mean transfer functions per cluster.

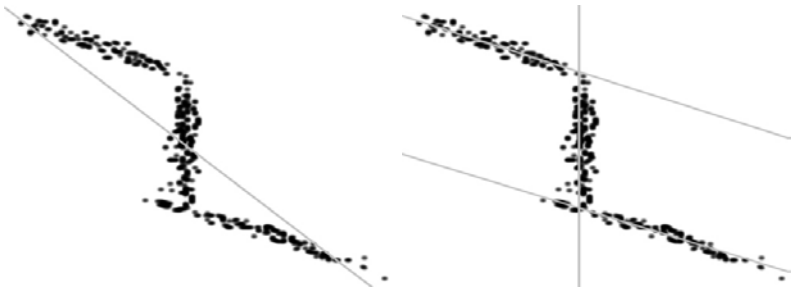


**Fig. 10.8: Difference between PCA (left) and local PCA (right): Clustering the data points captures local behavior and allows for more efficient or accurate compression.**

### 10.2.3 Non-Linear Approximation of Transfer Function and Lighting

Having computed the factorized representations of BTF and transfer, the third important step towards efficient PRT rendering is non-linear approximation of transfer and lighting. All-frequency PRT approach are commonly based on wavelet representations of both signals. Wavelets [500] are hierarchical functions representing scaled and translated copies of a locally defined waveform, called the *mother wavelet*. Due to their local behavior, they are suitable for representing both low-frequency and high-frequency signals. This property makes them an essential tool for signal processing, leading to widespread use in application areas like computer graphics, audio signal processing, and voice recognition. The specific type of wavelet used in this method, the Haar wavelet, was developed in 1909 by the Hungarian mathematician Alfréd Haar.

#### Definition

PRT rendering techniques usually assume distant lighting stored as a cube map, i.e., as a set of pixels. RGB cubemap images with $2^j$ pixels can be represented by three, piecewise constant functions $f_r$, $f_g$, $f_b$ which map the interval $[0, 1[$ onto $\mathbb{R}$ and which are all constant on subintervals $[0, \frac{1}{2^j}[$, $[\frac{1}{2^j}, \frac{2}{2^j}[$, ..., $[\frac{2^{j-1}}{2^j}, 1[$. If the vector space of these functions is denoted by $V^j$, a sequence of spaces results such that:

$$V^0 \subset V^1 \subset V^2 \subset \ldots$$

Functions can be represented in these vector spaces $V^j$ with arbitrarily small error if a suitable $j$ is chosen.

A natural basis of $V^j$ can be derived from the block functions

$$\phi_i^j(x) = \begin{cases} 1 & x \in \left[\frac{i}{2^j}, \frac{i+1}{2^j}\right[ \\ 0 & else \end{cases} \tag{10.4}$$

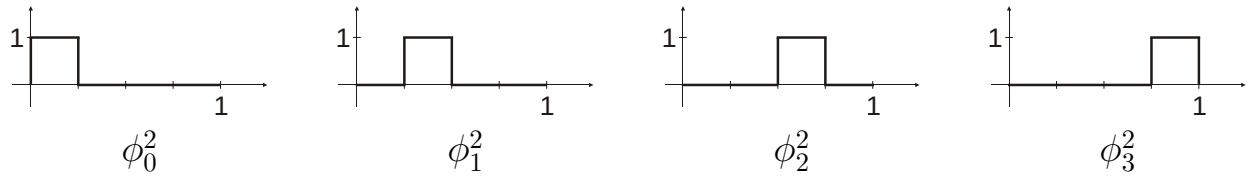



**Fig. 10.9: Block basis for representation of functions in $V^2$.**

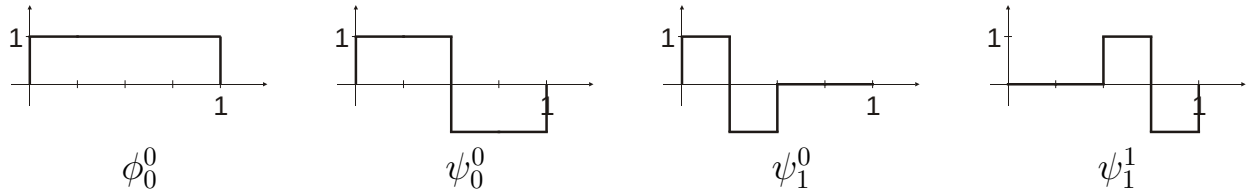



**Fig. 10.10: Haar wavelet basis for representation of functions in $V^2$.**

with $i = 0 \dots 2^j - 1$, which are depicted in Figure 10.9. A piecewise constant function $f$ can thus be represented in $V^j$ by $2^j$ coefficients $c_0^j, c_1^j, \dots, c_{2^j-1}^j$. Representing $f$ in $V^{j-1}$ results in a coarser approximation, yielding $2^{j-1}$ coefficients computed as average values $c_i^{j-1} = \frac{1}{2}\left(c_{2i}^j + c_{2i+1}^j\right)$ of the coefficients of the higher-resolution vector space $V^j$. Averaging introduces errors that cannot be represented in $V^{j-1}$. Instead, the errors are elements of the complementary vector space $W^{j-1}$ (note: $V^{j-1} \cup W^{j-1} = V^j$). The basis of such a vector space can be derived from the *Haar wavelets*

$$\psi_i^j = \begin{cases} 1 & x \in \left[\frac{i}{2^{j+1}}, \frac{i+\frac{1}{2}}{2^{j+1}}\right[ \\ -1 & x \in \left[\frac{i+\frac{1}{2}}{2^{j+1}}, \frac{i+1}{2^{j+1}}\right[ \\ 0 & else \end{cases}. \tag{10.5}$$

Thus, a function $f^{(j)} \in V^j$ can be represented by the coefficients for the basis functions of $V^{j-1}$ (the coarser representation $f^{(j-1)}$) and $W^{j-1}$ (the approximation error). Analogously to the $c_i^j$, the coefficients for wavelets $\psi_i^j$ can be computed as $d_i^{j-1} = \frac{1}{2}\left(c_{2i}^j - c_{2i+1}^j\right)$.

Repeated application yields the following orthogonal basis for a vector space $V^j$ (cf. Figure 10.10):

$$\phi_0^0, \psi_0^0, \psi_0^1, \psi_1^1, \psi_0^2, \psi_1^2, \psi_2^2, \psi_3^2, \dots, \psi_{2^j-1}^{j-1}.$$

To make the basis orthonormal, the scaling factors $\pm 1$ in the Equations 10.4 and 10.5 need to be replaced by $\pm 2^{j/2}$. Please note that the coefficient of $\phi_0^0$ represents an average value over all pixels. Therefore, $\phi_0^0$ is often referred to as *scale function*.

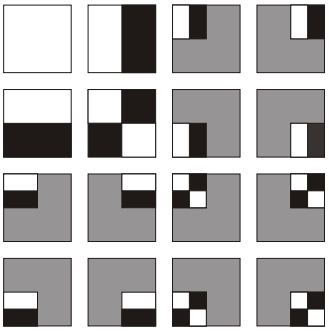


**Fig. 10.11: 2D non-standard Haar wavelet basis. White areas correspond to positive values, black to negative, and gray to zero-valued ones.**

## 2D Haar Wavelets

The basis defined above can be used for 2D images if images are linearized by concatenating pixels in either rows or columns. Unfortunately, in such a linearized version, pixels located next to each other in the image will become dislocated. Assuming that spatially close pixels feature a greater similarity than pixels located far from each other, this prohibits compact, accurate approximations of the images. Much more suitable results can be achieved by applying a basis of 2D wavelets, which is especially important for the following approximation of environmental light.

Let $f(x, y)$ a piecewise constant function with step size $2^n$ in both directions. Representing $f$ by 2D wavelets is possible in two ways: The so called *standard* basis is derived from all possible tensor products of two 1D wavelets. Therefore, the fineness of induced wavelets, which is defined by $j$, is usually different for $x$ and $y$, which is often not desired. Thus, the *non-standard* 2D wavelet basis is more commonly used, which is composed of the scale function $\phi_0^0(x)\phi_0^0(y)$ and the wavelets $\phi_l^j(x)\psi_k^j(y)$, $\psi_l^j(x)\phi_k^j(y)$, and $\psi_l^j(x)\psi_k^j(y)$, for $j = 0 \ldots n-1$ and $l, k = 0, \ldots, 2^j - 1$. The basis is orthonormal if orthonormal 1D basis functions are used. A simple graphical visualization of the non-standard 2D basis is depicted in Figure 10.11.

## Approximation

To represent a spherical function by 2D wavelet coefficients, a suitable 2D parameterization of the sphere has to be chosen. On possible parameterization is the cube map, which requires that all six sides of the cube be represented individually. If only the first $N$ basis functions are used – such an approximation is called *linear* in the following – and if $N$ is relatively small, then non-local bases like spherical harmonics lead to better results than Haar wavelets. Yet, if more basis functions are employed, then wavelets preserve details much better.

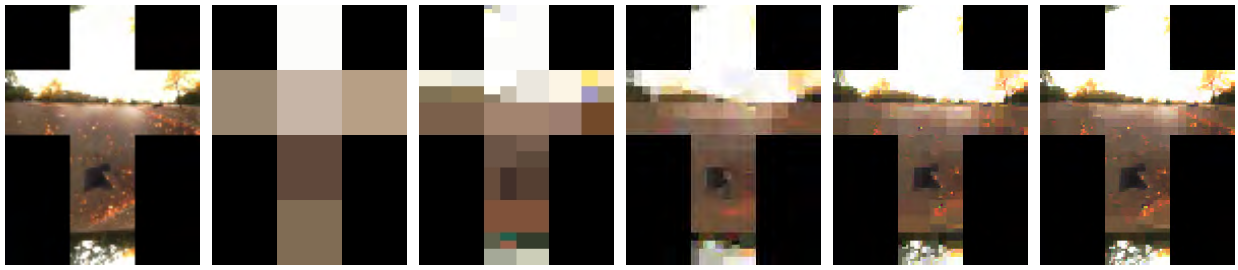



**Fig. 10.12: Non-linear approximation of a cube map lighting environment with Haar wavelets. The number of coefficients refers to each side of the cube map.**

To take advantage of this property, a so-called *non-linear* approximation is applied: When processing wavelet coefficients as in Equation 10.2, wavelets from all levels are handled but all coefficients but the ones with the $N$ highest absolute values are set to zero. For a typical cube map, most of the coefficients are very small. Therefore, only small errors result from setting these coefficients to zero, and additionally, these errors have local influence only. As a result of this, typically very few coefficients (0.1% to 1%) suffice for accurate approximations of lighting environments which can represent area light sources (sky or windows) as well as point light sources, which correspond to a single pixel in the cube map (cf. Figure 10.12).

Like the incident lighting, the location independent part $q$ of the transfer function is approximated non-linearly by Haar wavelets. Combining light and transfer therefore requires a dot products of high-dimensional vectors, but fortunately most of the vector components are zero. Employing specialized storage and computation methods for sparse vectors leads to storage and processing costs proportional to $N$, and thus independence of the size of the cube map and the transfer functions.

### 10.2.4 Rendering

The PRT representation derived above gives rise to efficient rendering on existing GPUs using shader programs. Equation 10.2 consists of four main parts, each corresponding to one of the sums in the equation.

- The most inner sum of light-dependent transfer components $q$ and lighting $L_i$ has to be evaluated whenever lighting changes (i.e., whenever the object is rotated w.r.t. the lighting environment or whenever the lighting environment is changed itself). The necessary computations are independent of particular vertices or fragments and are thus most efficiently performed on the CPU.

- The sum of the spatially dependent transfer components $p$ and the most inner

sums, which has to be computed for every vertex of the rendered mesh, is most naturally computed in vertex shaders.

- Interpolation over the three closest vertices can be performed using fixed-function graphics hardware.
- Summation of the spatially-dependent BTF parts and remaining sums needs to be executed for every fragment and is therefore implemented as a fragment shader.

In the following, brief descriptions of the four evaluation stages are given. The approach assumes Shader Model 3.0 [393] since it requires texture lookups in vertex shaders and loops both in vertex and fragment shaders.

### Computations on the CPU

Evaluating the sum

$$s_{\mathbf{v},c,l,k(\mathbf{t}_d,\mathbf{v},c)} = \sum_{\mathbf{l}_j} q_{\mathbf{v},c,l,k(\mathbf{t}_d,\mathbf{v},c)}\left(\mathbf{l}_j\right) L_i(\mathbf{l}_j)$$

of products between transfer function components and lighting requires evaluation of high-dimensional inner products. Typically, the vectors contain several thousand coefficients. Fortunately, this large number can be reduced significantly by non-linear approximation with Haar wavelets as outlined above. The approximated vectors still contain several thousand entries but only few of them do not equal zero, representing sparsely occupied vectors. Efficient computations on such vectors are possible with libraries like $\mu$BLAS [41]. Results from this step are made accessible to the following stages by storing them as textures.

Vertex and fragment shaders additionally require access to the indices and interpolation weights of the closest local view directions. It is therefore convenient to precompute these on the CPU, to store them in a parabolic map [216] texture, and to make them available to the shaders. The parabolic map representation is chosen due to almost equal sampling density and since computation of texture coordinates can be performed efficiently.

### Vertex Shader

The vertex shaders needs to compute the sum of products of the weights $p$ and the precomputed sums $s$ for each transfer component $c_T$. To implement this, in a first step the local view direction is computed from the eye position, the vertex location, and the

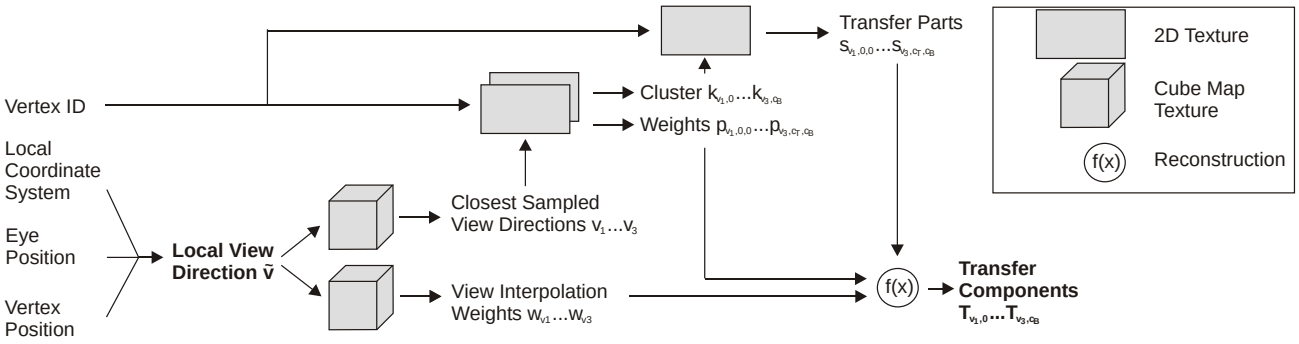



**Fig. 10.13: Schematic description of PRT vertex shader. Entities written in bold font are passed to the fragment shader.**

local coordinate system. With one texture lookup each, the indices and interpolation weights of the three closest BTF sample directions can be determined. For each of these sampled directions and each BTF component $c_B$ the cluster index $k(\mathbf{t}_d, \mathbf{v}, c_B)$ is read from texture memory. Finally, for each component $c_T$ the reconstruction weights $p$ and precomputed sums $s$ are looked up, and the weighted sum is computed. Figure 10.13 visualizes this in a simple schematic form.

The total costs for the shader consist of two texture lookups to determine indices and interpolation weights of the three closest view directions, $3(C_B + 1)$ lookups for the clusters, and $3(C_B + 1)(C_T + 1)$ for the weights and precomputed sums each. For a typical configuration with $C_T = 2$ this requires 65 texture accesses already.

Results of the vertex shader are passed to the fixed function GPU pipeline, which interpolates the per-vertex data for every fragment to be rendered. The interpolated values serve as input to the following fragment shader.

## Fragment Shader

Compared to the vertex shader, the fragment shader is relatively simple and short. For each component $c_B$ of the BTF factorization and each of the closest sampled view directions $v_i$, the respective Eigen-Texture $h_{v_i,c_B}$ needs to be combined with the interpolated results $T_{v_i,c_B}$ from the vertex shader (a schematic view of the shader is shown in Figure 10.14). The closest view directions and the respective interpolation weights are determined from the interpolated local view direction by accessing two cube maps as in the vertex shader. Based on these values and the texture coordinates of the fragment, the corresponding pixels from the Eigen-Textures are looked up of from a 3D texture. Finally, the fragment color is computed as the interpolated sum of products of Eigen-Textures and vertex shader results. The final operations requires an additional $3(C_B+1)$ texture accesses.

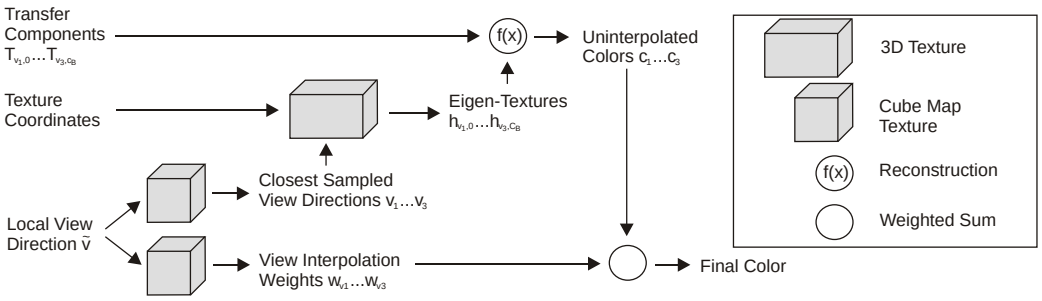



**Fig. 10.14: Schematic description of PRT fragment shader.**

Depending on the dynamic range of the BTF data and the lighting, it is necessary to perform a tone mapping of the fragment color to transform the HDR color to the displayable range. The images shown in this section were created by scaling with an appropriate factor and gamma correcting the products. Obviously, more correct results can be achieved in combination with more involved tone mapping methods.

### 10.2.5 Results

The PRT approach presented in this chapter retains the distinction between geometry and material. Therefore, it is especially suitable for low-resolution meshes since small and medium scale surface details can be encoded as materials. The following measurements were made with a coarse model of a shirt with 3.4k vertices. Several BTF materials were chosen to cover the shirt, among them the cushion material (cf. top left image in Figure 10.15) which features a spatial resolution of 120 $\times$ 129 texels and a directional resolution of 151 measured view and light directions each. Other BTFs like the knitwear (cf. bottom images in Figure 10.15) consist of 256 $\times$ 256 texels and were sampled at 81 view and light directions each. The BTFs were factorized as described above, retaining only the mean value and the two most significant components.

As per Equation 10.1 the transfer function $T$ depends on view direction $\mathbf{v}$, BTF component $c_B$, vertex $\mathbf{t}$, and light direction $\mathbf{l}_j$. For the shirt model and a resolution of $6\times32\times32$ for the HDR environment map this leads to a total amount of 35 GB of data that needs to be factorized. To handle this huge amount of data, the algorithm iterates between evaluation for one view direction and factorization of computed data using local PCA.

Due to the extreme size of data sets and the costs for factorization the time requirements of the approach are very high. Even low environment map resolutions like $6\times8\times8$ pixels lead to run times of about six hours for the cushion BTF and about three hours for the knitwear BTF, which are mainly due to local PCA. Such low resolutions

**Fig. 10.15: Sweater in building environment covered with different kinds of materials.**

match the directional resolutions of BTFs and should thus be sufficient for reflections (cf. Figure 10.16). Yet, higher resolutions are required to simulate complex shadows. For $6\times32\times32$, run times reach about 64 hours for the cushion BTF and about 32 for the knitwear BTF. Since evaluations are performed per view direction, parallel evaluation on several CPUs is trivially possible.

Like preprocessing time, storage requirements of the compressed PRT solution are resolution dependent: For six transfer components and $6\times8\times8$ light directions, about 122 MB are required. For $6\times32\times32$ light directions, this increases to about 700 MB. In addition, the factorized BTFs need to be stored, which requires about 24 MB for the cushion material. Due to separate storage, they can be reused for different objects.

The significant memory requirements lead to restrictions for real time rendering since all data needs to be made available to the GPU in textures. The optimal light resolution therefore also depends on the graphics card memory (currently: up to 1 GB) and the maximum size of textures (currently: $4096^2$ texels). Especially problematic in this context are the cluster indices $k_T$ and the transfer function reconstruction weights $p$, since

8×8, 40 coeff 4×4, 16 coeff 2×2, 4 coeff

**Fig. 10.16: Reduction of rendering quality due to reduced resolution of lighting. Differences become imperceptible for lighting resolutions of $8^2$ and $4^2$ per cube map side, which is in the range of the angular BTF resolution of the relatively diffuse material.**

both depend on the number of vertices $|\mathcal{V}|$ of the model, the number of view directions $|M_r|$, and the number of retained BTF components $C_B$. The number of indices is given by $|\mathcal{V}|(C_B + 1)|M_r|$, the number of weights $|\mathcal{V}|(C_B + 1)(C_T + 1)|M_r|$ is even larger since it additionally depends on the number of transfer components $C_T$. As a result, if 2 BTF and 6 transfer components are used, the number of vertices that can efficiently be stored in a $4096^2$ RGBA texture is 16k. For meshes with higher resolution, multiple textures have to be used.

For the shirt model covered with the cushion material, about 41 MB are required for the texture holding the weights. For more complex meshes memory requirements can be eased by employing 16 bit float values, which reduces the accuracy of rendered images.

Further data stored in textures are the indices and weights for interpolating closest view directions and the precomputed sums $s$. The size of both is not related to either the resolution of the mesh or the number of light directions, and they require significantly less memory than the previous textures – less than two MB each.

The rendering speed of the proposed PRT solution is quite interactive but depends on the number of employed BTF and transfer components. For two BTF components and six transfer components, which leads to high quality images as shown in Figure 10.15, about 20 fps are achieved for static light situations on a nVIDIA GeForce 7800 FX graphics card at a resolution of 1280×1024 pixels. For four and two transfer components, frame rates increase to 27 fps and 32 fps at the cost of reduced display quality (cf. Figure 10.17). Since the vertex shader performs most of the work, display rates are almost independent of screen resolution which is highly desirable for industrial use.

Changing the lighting environment by either rotating or exchanging it requires updating the precomputed sums $s$. Depending on the number of wavelet coefficients rep-

4 components 2 components 1 component

**Fig. 10.17: Reduction of rendering quality due to reduced number of transfer components. For the shirt model, hardly any differences are visible between the 4 and 2 component versions.**

resenting each side of the cube map, the number of transfer clusters, the number of BTF and transfer components, and the angular resolution of the BTF, this process takes some seconds. For a typical high-quality scenario with six transfer components, two BTF components, 32 transfer clusters, and 81 sampled light directions in the BTF, 0.5 seconds are required if 20 wavelet coefficients are retained per cube map side. The number increases linearly in all parameters but remains low for typical applications. Nevertheless, changes to the lighting environment cannot be performed in real-time.

### 10.2.6 Conclusions

The PRT approach presented above is the first approach that combines all-frequency PRT effects and complex, spatially varying materials. It therefore represents a natural extension of the approaches of Sloan et al. [488] and Müller et al. [370] which are limited to low-frequency lighting effects. The novel approach achieves about the same frame rates as previous approaches due to the efficient use of vertex- and fragment shaders. Especially interesting for industrial users, which require high-resolution display, is that its performance is almost independent of display resolution.

Experiments with the novel technique show that all-frequency lighting is not sufficient to produce perfect quality results. Figure 10.16 demonstrates that increasing the lighting resolution beyond the angular resolution of the material BTF does not lead to any further improvements in rendering quality. For such modest light resolutions, preprocessing times and storage requirements of this method are quite acceptable. The efforts rise substantially for higher resolutions, yet, it does not appear useful to employ the proposed method for simulating effects other than reflection. For all-frequency

shadowing effects, which are neither view- nor material-dependent phenomena, the presented technique is overly complex. Therefore, simpler and more efficient techniques like the all-frequency PRT approach of Ng et al. [382] or real-time soft shadowing techniques [205] should be used. For almost specular materials which require much higher sampling rates, PRT approaches need to store a significant amount of memory and are thus not very useful either.

Unlike spherical harmonics, which represent functions on the sphere and are efficiently rotatable, 2D wavelets need to encode the six sides of an environment map separately. Therefore, efficient rotation of wavelet coefficients is not possible in an analytical way. As a result, all-frequency PRT techniques based on wavelets do not support efficient rotation of lighting environments. Recently, Wang et al. [558] proposed an approach for solving this problem based on precomputing a set of fixed wavelet rotations. Unfortunately, good results can only be achieved for a large set of precomputed rotations, which leads to tremendous increases of memory requirements.

Another significant drawback of all-frequency PRT approaches and this one in special is the large amount of memory that needs to be computed, stored, and used for real-time rendering. While it seems difficult to reduce the amount of computed data, the amount of data used for rendering could significantly be reduced if the per-view factorization of the BTF would be replaced by local PCA compression. As mentioned above, this approach was not taken due to the large amount of memory that needs to be processed at once. Fortunately, this problem can be resolved using incremental PCA techniques [46] since new values can be added to the data basis on the fly. Such an approach would reduce the necessary storage requirements for weights from $|V|(C_B+1)(C_T+1)|M_r|$ to $|V|(C_B+1)(C_T+1)|k_B|$ where $|k_B| \leq 8$. Thus, storage requirements would be reduced by at least an factor of 10 to 40, depending on the number of BTF components $|k_B|$ and the angular resolution of BTFs. Use of incremental PCA would additionally allow encoding more complex meshes. Currently at most 16k vertices can be handled, which would increase to about 100k to 400k. Thus, models with industrially relevant complexity would become possible.

Another improvement left for future work is combination with other existing PRT techniques that allow for handling of more specular materials, deformable scenes, and local lighting. Due to the large number of PRT extensions published so far (cf. Chapter 8) this tasks appears like a very promising yet labor-intense task, especially since most techniques feature their own limitations. Thus, combinations of techniques need to make sure that they combine the strengths of the techniques and not their weaknesses.

## 10.3 Discussion

In this chapter two different techniques for real-time rendering of global illumination effects from precomputed data were presented. Both start with precomputing (parts of) the Rendering Equation, compress the huge amounts of resulting data, and render them at real-time frame rates.

As shown above, SLF rendering leads to very high quality images, with the quality of the results only limited by the resolution of SLFs. For glossy-diffuse materials, mainly the spatial resolution is the limiting factor, especially it objects with large surfaces are handled. Unfortunately, SLFs prohibit interactive exchange of geometry, lighting, and material, despite the improvements proposed above.

In contrast, PRT rendering allows for efficient exchange of distant lighting. Modifications to materials or the geometry are not possible with the presented approach but might be enabled in combination with existing techniques. Unlike SLFs, the quality of PRT rendering techniques is limited by the resolution of lighting, of the BTF, and of the mesh. Since the presented technique retains the separation of geometry and material, typically less data needs to be precomputed and stored than for SLFs, especially if large objects with uniform materials are used. Unfortunately, the presented PRT technique prohibits effects due to non-distant light sources, which tend to add important detail.

In summary, in the context of predictive VR SLF rendering is more suitable for high-quality inspection of models in selected scenarios where neither the model nor the surrounding environment need to be changed. In contrast, PRT should be employed if model variants are checked in varying lighting environments.

Unfortunately, none of the presented techniques is currently capable of producing predictive renderings, just like all other real-time rendering approaches. Reasons for this shortcoming are manifold. Obviously, the accuracy of rendered scenes is insufficient. Additionally, approaches based on precomputed data require huge amounts of memory to store all necessary data, especially if mixtures of low-frequency and high-frequency content need to be handled. The presented approaches succeed at producing nearly photorealistic results for diffuse and glossy-diffuse materials and all-frequency illumination. Nevertheless, they fail to handle surfaces covered with highly specular materials. Adding reflections in such surfaces is the topic of the next chapter.

# Chapter 11

# Real-Time Reflections

As the previous chapters showed the degree of realism achieved by rendered images largely depends on the inclusion of global lighting effects like shadows and light exchange between surfaces. Chapter 10 presented methods that account for these effects but are limited to glossy-diffuse materials. Therefore, in order to simulate light exchange in scenes containing highly reflective materials like mirrors, polished metals, lacquers, or glass additional methods are required.

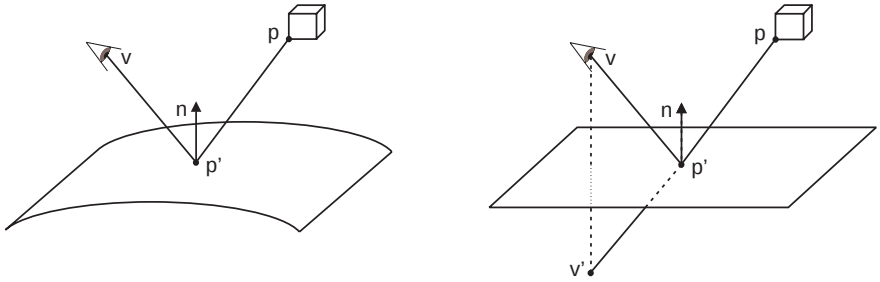
When computing such light exchange, two types of interactions need to be distinguished: reflections and refractions. In many scenes refractions can safely be omitted since most materials show no transparency for reasonable thickness. If a scene contains an object consisting of such a material which is thin enough to show transparency, refraction effects can typically be neglected (this is not the case for objects like lenses).

In contrast, reflections often play an important role since many every-day objects like mirrors, windows, glasses, bottles, CDs, TV screens and many more show such effects. While simulation of reflections in planar surfaces represents an easy task, the approach becomes much more complex for curved surfaces. In the following three approaches for computing reflections in curved surfaces are presented.

## 11.1 Analytic Reflections for Bézier Surfaces

Rendering reflections in planar reflectors is relatively simple and efficient using graphics hardware (cf. Figure 11.1(b)): The viewpoint $\mathbf{v}$ is mirrored in the reflecting plane $P$ (with normal $\mathbf{n_p}$ and distance to origin $d_p$) and the scene is rendered whereas projection points $\mathbf{p}'$ for vertices $\mathbf{p}$ are determined by solving the equation

$$\mathbf{p}' = \mathbf{v}' + \frac{d_p - \mathbf{n_p} \cdot \mathbf{v}'}{(\mathbf{p} - \mathbf{v}')\mathbf{n_p}}(\mathbf{p} - \mathbf{v}') \tag{11.1}$$




(a) General surface $S$ (b) Planar surface $P$

**Fig. 11.1: Perfect reflections for various types of surfaces.**

resulting from intersecting plane $\mathbf{n_p} \cdot \mathbf{x} = d_p$ and line $\mathbf{x} = \mathbf{v}' + \alpha(\mathbf{p} - \mathbf{v}')$.

Unfortunately, such simple approaches are not available for curved reflectors: Analytic solutions require solutions to non-linear equations. Nevertheless, due to the increasing power of GPUs computing such solutions became feasible in the last years (see [202] for a recent overview). In the following such an approach for simulating reflections in curved reflectors is presented and discussed.

### 11.1.1 Approach

The problem of computing reflections on a surface $S$ can be stated as follows: Given a viewpoint $\mathbf{v}$ and a point $\mathbf{p}$ in 3D space, one wants to determine the point $\mathbf{p}' \in S$ (or the points $\mathbf{p_i'} \in S$) such that the law of perfect reflection

$$\mathbf{p}_n \;=\; -\mathbf{v}_n + 2 \cdot \left(\mathbf{v}_n^T \mathbf{n}'\right) \cdot \mathbf{n}' \tag{11.2}$$

holds, with $\mathbf{p}_n = \frac{\mathbf{p}-\mathbf{p_i'}}{||\mathbf{p}-\mathbf{p_i'}||}$ being the normalized object direction, $\mathbf{v}_n = \frac{\mathbf{v}-\mathbf{p_i'}}{||\mathbf{v}-\mathbf{p_i'}||}$ the normalized view direction, and $\mathbf{n}'$ the normal of $S$ in point $\mathbf{p}'$ (cf. Figure 11.1(a)).

Determining $\mathbf{p}'$ by solving Equation 11.2 directly unfortunately leads to highly complicated, numerically problematic computations due to the required normalization of object and view direction. Fortunately a very basic law of geometric optics saves the day: Fermat's Principle [199]. The principle states that light traveling from one point to another will always travel along paths of stationary optical lengths, i.e., along paths with locally maximal or minimal lengths. This leads to the much simpler equation

$$d \;=\; ||\mathbf{p}' - \mathbf{v}|| + ||\mathbf{p}' - \mathbf{p}||. \tag{11.3}$$

The point $\mathbf{p}'$ (or in general the points $\mathbf{p_i'}$) can therefore be located by determining the locally extreme path length (or lengths) $d$. For the case of convex reflectors, it obviously

follows that exactly one minimum exists. In addition no inflexion points exist, which greatly simplifies the solving process.

Although this approach is applicable for every type of surface, the approach presented here is specialized for convex Bézier surfaces [138]. Reasons for this specialization are the high industrial relevance of this kind of surface and the reduced complexity when assuming convex surfaces.

### 11.1.2 Bézier Surfaces

Among all the various representations for general surfaces Bézier surfaces are one of the most commonly used, especially since every Non-Uniform Rational B-Spline (NURBS) surface can be subdivided into a set of them [138]. Bézier Surfaces

$$S(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} \mathbf{b}_{ij} B_i^m(u) B_j^n(v) \tag{11.4}$$

are parametric surfaces and generally defined over the interval $[0,1]^2$. $n$ and $m$ denote the degree of the surface, the $\mathbf{b}_{ij} \in \mathbb{R}^3$ denote control points that determine the shape of the surface. The definition is based on the Bernstein-Polynomials

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}, i = 0, 1, ..., n \tag{11.5}$$

where $n$ denotes the degree of the polynomial.

Bézier surfaces have various helpful properties for modeling and computation [138], among them the property that the derivatives of them form other Bézier surfaces with reduced degree. This leads to a uniform evaluation scheme for surface points and derivatives. An additional nice property concerning efficient evaluation is that Equation 11.4 can be written in matrix form:

$$S(u,v) = [u^0 \dots u^m]\, M^T \begin{bmatrix} \mathbf{b}_{00} & \cdots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \cdots & \mathbf{b}_{mn} \end{bmatrix} N \begin{bmatrix} v^0 \\ \vdots \\ v^n \end{bmatrix} \tag{11.6}$$

where $M_{ij} = (-1)^{j-1} \binom{m}{j} \binom{j}{i}$ and $N_{ij} = (-1)^{j-1} \binom{n}{j} \binom{j}{i}$.

### 11.1.3 Reflections in Bézier Surfaces

For the specific case of a convex Bézier surface Equation 11.3 changes to

$$d(u,v) = ||S(u,v) - \mathbf{v}|| + ||S(u,v) - \mathbf{p}|| \tag{11.7}$$

and the problem changes to determining parameter values $(u, v)$ such that $d(u, v)$ is minimized. Such problems are commonly tackled using minimization algorithms that especially suite the task, i.e., the kind of function and different constraints. Since minimization should be performed on the GPU in real-time such constraints are

- fast convergence,
- limited number of instructions for coding the algorithm, and
- limited number of executed instructions.

The first point needs obviously be fulfilled for every kind of real-time algorithm. The second and third points stem from graphics hardware restrictions which might be removed in the future. In order to conform to the restrictions, a Newton-Raphson-solver [422] was chosen for implementation since it promises quick convergence by using second order derivatives. Formulas for computing these derivatives are as follows:

$$
\begin{aligned}
\frac{\partial d(u,v)}{\partial u} &= \left(\frac{\partial S(u,v)}{\partial u}\right)^T \left(\frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||} + \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||}\right) \\
\frac{\partial d(u,v)}{\partial v} &= \left(\frac{\partial S(u,v)}{\partial v}\right)^T \left(\frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||} + \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||}\right) \\
\frac{\partial^2 d(u,v)}{\partial u^2} &= \left(\frac{\partial^2 S(u,v)}{\partial u^2}\right)^T \left(\frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||} + \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||}\right) + \\
&\quad \left(\frac{\partial S(u,v)}{\partial u}\right)^T \left(\frac{1}{||S(u,v)-\mathbf{v}||}\left(\frac{\partial S(u,v)}{\partial u} - \frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||^2}\left(S(u,v)-\mathbf{v}\right)^T \frac{\partial S(u,v)}{\partial u}\right) + \right. \\
&\quad \left. \frac{1}{||S(u,v)-\mathbf{p}||}\left(\frac{\partial S(u,v)}{\partial u} - \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||^2}\left(S(u,v)-\mathbf{p}\right)^T \frac{\partial S(u,v)}{\partial u}\right)\right) \\
\frac{\partial^2 d(u,v)}{\partial u \partial v} &= \left(\frac{\partial^2 S(u,v)}{\partial u \partial v}\right)^T \left(\frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||} + \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||}\right) + \\
&\quad \left(\frac{\partial S(u,v)}{\partial u}\right)^T \left(\frac{1}{||S(u,v)-\mathbf{v}||}\left(\frac{\partial S(u,v)}{\partial v} - \frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||^2}\left(S(u,v)-\mathbf{v}\right)^T \frac{\partial S(u,v)}{\partial v}\right) + \right. \\
&\quad \left. \frac{1}{||S(u,v)-\mathbf{p}||}\left(\frac{\partial S(u,v)}{\partial v} - \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||^2}\left(S(u,v)-\mathbf{p}\right)^T \frac{\partial S(u,v)}{\partial v}\right)\right) \\
\frac{\partial^2 d(u,v)}{\partial v^2} &= \left(\frac{\partial^2 S(u,v)}{\partial v^2}\right)^T \left(\frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||} + \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||}\right) + \\
&\quad \left(\frac{\partial S(u,v)}{\partial v}\right)^T \left(\frac{1}{||S(u,v)-\mathbf{v}||}\left(\frac{\partial S(u,v)}{\partial v} - \frac{S(u,v)-\mathbf{v}}{||S(u,v)-\mathbf{v}||^2}\left(S(u,v)-\mathbf{v}\right)^T \frac{\partial S(u,v)}{\partial v}\right) + \right. \\
&\quad \left. \frac{1}{||S(u,v)-\mathbf{p}||}\left(\frac{\partial S(u,v)}{\partial v} - \frac{S(u,v)-\mathbf{p}}{||S(u,v)-\mathbf{p}||^2}\left(S(u,v)-\mathbf{p}\right)^T \frac{\partial S(u,v)}{\partial v}\right)\right)
\end{aligned}
$$

Despite the lengthy expressions required to compute derivatives, the terms can be evaluated very efficiently since they are composed of a small number of repeating subterms.

**Procedure 2** renderBezierReflections

```
render the scene without reflectors
for all reflectors r do
    clear stencil buffer
    render r into stencil buffer
    activate reflector vertex program
    pass evaluation matrices as local vertex program parameters
    render scene geometry
    deactivate reflector vertex program
end for
```

When applying the solver to Equation 11.7 one usually experiences convergence within a very few number of steps when starting at parameter value $(u, v) = (0.5, 0.5)$. Even vertices projected to positions corresponding to parameter values outside of the intended domain $[0, 1]^2$ lead to quickly converging solutions. Please note that projecting these vertices is necessary in general since – although they themselves will not be visible in the reflector – incident triangles might be.

For some configurations, evaluation of $S(0.5, 0.5)$ and the corresponding derivatives suggests a far too wide step leaving the intended domain. Since Bézier surfaces that are convex with regard to the parameter domain $[0, 1]^2$ need not be convex for different parameter values at all, this might lead to situations where the projected position converges against an unintended extremum. The standard solver was therefore enhanced by a simple line search algorithm: After determining the step $(u_s, v_s)$ for current parameter values $(u, v)$ the algorithm tests whether $d(u+\frac{1}{2}u_s, v+\frac{1}{2}v_s) < d(u+u_s, v+v_s)$ in which case the step size is halved and the test is repeated until it fails. In all tested cases this simple additional test prevented unintended extrema, avoiding disturbing artifacts due to *jumping* vertices and thus reflected objects with almost arbitrarily changing shape.

### 11.1.4 Rendering Algorithm

In order to render specular reflections the pseudocode from Procedure 2 is executed. In the first step, the scene without specular reflectors is rendered. Afterwards, for every reflector first the stencil buffer is activated and reset and then the geometry of the reflector is rendered into this buffer in order to avoid overdraw of pixels that are not part of the reflector during the following steps. Next, the reflector vertex program which contains the implementation of the enhanced Newton-Raphson solver is activated and the necessary parameters (i.e., the matrices necessary for evaluation of surface points, first and second derivatives) are handed to it. Finally, the scene geometry is rerendered which now results in reflections on the currently active reflector.

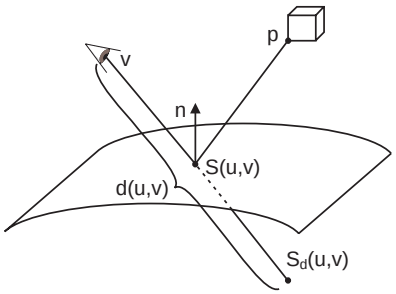



**Fig. 11.2: Construction of the depth-corrected reflection point for rendering.**

Directly using the reflection point $S(u, v)$ from the solver as new position for the rendered vertex $\mathbf{p}$ leads to serious aliasing effects since the complete scene is flattened and therefore correct occlusion cannot be achieved. Therefore the vertex $\mathbf{p}$ is not moved to $S(u, v)$ but to the more distant point $S_d(u, v)$ by pushing back $S(u, v)$ along the line of sight until the distance to the viewpoint equals $d(u, v)$. To achieve this, the rendering algorithm first transforms the viewpoint $\mathbf{v}$ to object space, then computes the vector $\mathbf{w} = S(u, v) - \mathbf{v}$ and finally computes the depth-corrected reflection point $S_d(u, v) = \mathbf{v} + d(u, v)\mathbf{w}$ (see Figure 11.2).

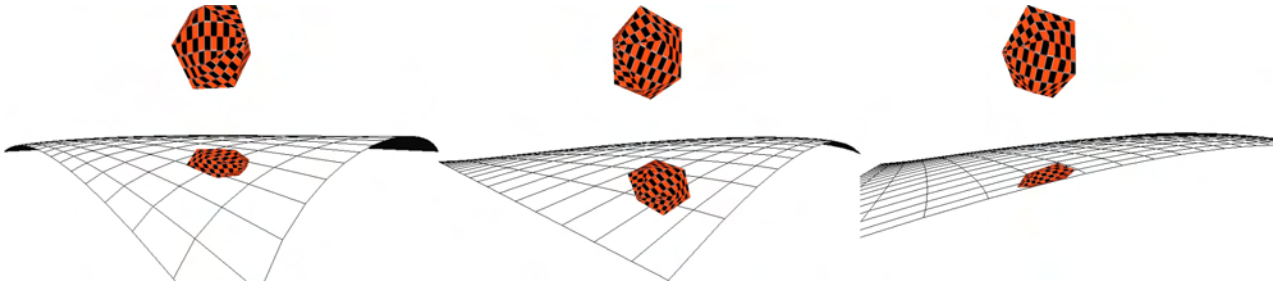
To incorporate this into the rendering algorithm, two steps have to be added. After rendering $r$ into the stencil buffer, the depth values of visible pixels corresponding to $r$ have to be set to the maximum depth. After rendering the scene geometry, the depth value of these pixels has to be set to the depth value of the reflector geometry.

Figure 11.3 shows screenshots from the program implementing the rendering algorithm. Textured models are reflected in a curved bicubic Bézier surface. Determining the position of reflected vertices requires a very small number of iterations, typically between two and five.

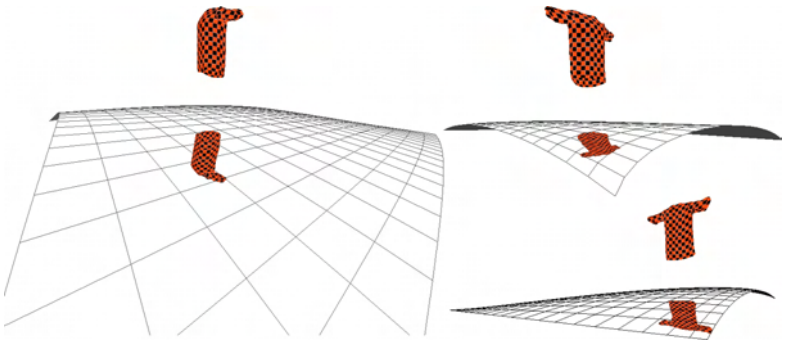
### 11.1.5 Discussion of Results

The approach for computing reflections in curved Bézier surfaces gives good results for a number of cases (some of them shown above). Unfortunately, the approach has various limitations and problems as well.

First, the presented approach is limited to convex surfaces. For concave reflectors vertices may project to multiple positions, which cannot be handled by current real-time graphics pipelines and which requires more complex solvers for the minimization process. The problem can be simplified by subdividing convex surfaces if necessary, nevertheless, detection of such cases is clearly non-trivial. In fact, for special configurations vertices may project to an infinite number of positions. Resolving such situations appears to be very difficult.



(a) Icosahedron



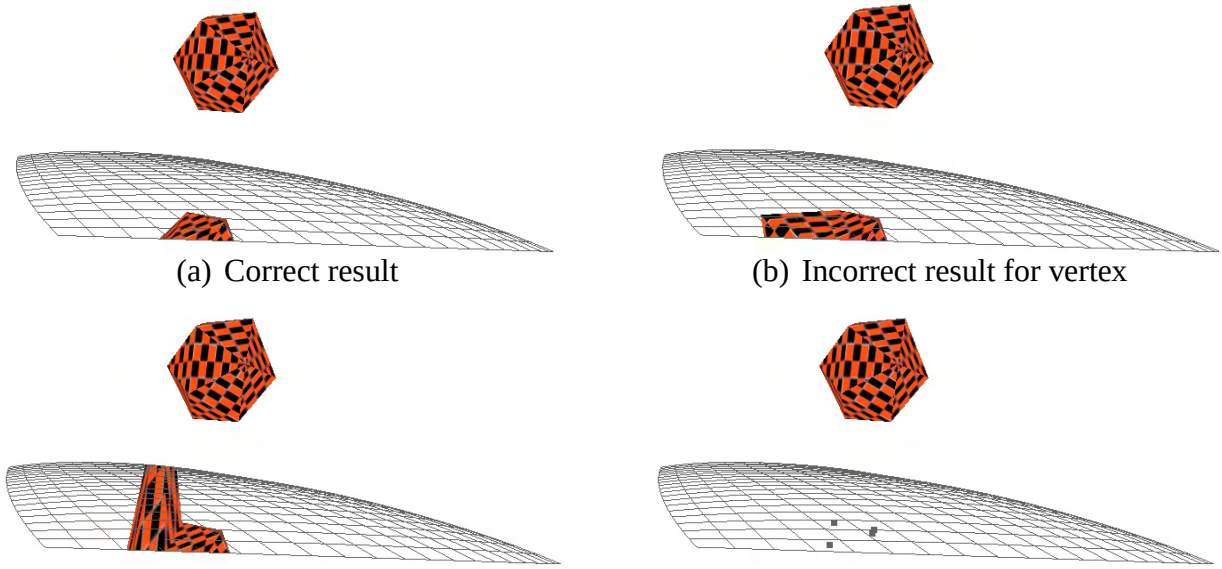
(b) Sweatshirt

**Fig. 11.3: Successful simulations of reflections in a curved surface.**

Second, the approach works nicely for relatively flat surfaces only. For regions with high curvature the optimization algorithm requires much higher numbers of iterations, thereby reducing rendering performance or even prohibiting execution on GPUs. Limiting the number of iterations then leads to errors as visible in Figure 11.4(b), where one vertex position did not converge to the global minimum.

Third, problems may occur when leaving the intended parameter domain $[0, 1]^2$ for evaluating Bézier surfaces. Even convex surfaces show concave behavior in such regions, leading to unpredictable results and thus convergence to unintended minima. Although vertices not projecting onto the reflector are not visible in the rendered image, they still cause serious problems which are shown in Figure 11.4(c). The position of one of the reflected vertices lies outside of the reflecting surface in an unintended minimum. While the vertex is invisible itself, the fragments resulting from assembling triangles from projected vertices are clearly visible, causing disturbing errors. Figure 11.4(d) shows that such problems can be omitted when rendering points only (points are draw for the vertices only). Therefore, point-based rendering methods might still find this approach useful.

Fourth, the process of assembling triangles from projected vertices is not correct since lines connecting the unprojected vertices should appear being curved after projection in most configurations.



(a) Correct result (b) Incorrect result for vertex

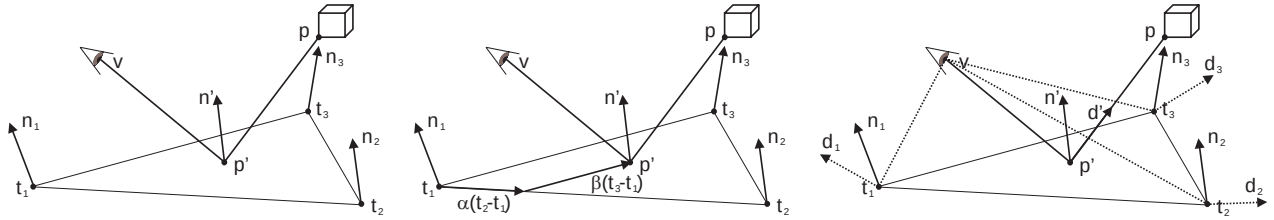
(c) Incorrect result due to interpolation (d) Correct placement of vertices

**Fig. 11.4: Problems when computing reflections in bicubic Bézier surface. Slight movements of the viewpoint lead to a number of correct and incorrect results.**

Fifth, the rendering algorithm as presented above requires the complete scene geometry to be rendered once for every reflector. Fortunately, the resulting overhead can be minimized employing spatial data structures (e.g., in combination with normal cones [479] large parts of the scene can be culled away especially for approximately flat reflectors).

In summary, the presented approach represents an interesting solution to the problem, based on a very simple principle. Unfortunately, application of this approach to Bézier surfaces leads to many difficulties, thus achieving correct results in real-time for a small number of configurations only.

Some of these problems were resolved by similar approaches developed concurrently. Estalella et al. [132] proposed to resample the positions and normals of all surfaces of a star-shaped object into a single cube map and determining reflection points for this representation. This reduces rendering costs since the surrounding scene has to be rendered only once for the whole object at the cost of restricting object topology and limiting accuracy to the cube-map resolution. Yet, their approach has another significant advantage: Closed objects are represented as a whole and thus feature no borders. The borders, or more specifically, the undefined behavior for parameter values outside the intended domain of Bézier surfaces, caused several problems in the presented approach. These can be omitted when merging surfaces. Unfortunately, the presented CPU [132] and GPU [131, 440] implementations still seem to be limited to rather smooth surfaces since results were presented for almost spherical objects only.




(a) Normals are interpolated over the triangle.

(b) Barycentric coordinates $(\alpha, \beta)$.

(c) Reflected vectors can be interpolated as well.

**Fig. 11.5: Reflections in triangle with per-vertex normals.**

# 11.2 Analytic Reflections for Triangular Surfaces

Most of the limitations and problems of the previous approach stem from the complexity of Bézier surfaces. Due to the appeal of the simplicity of the underlying principle (i.e., projection onto the surface according to some formula) another approach for a simpler surface representation (i.e., triangle meshes) was developed. The approach was inspired by the work of Yu and Millan on Multiperspective Rendering [244], which was later employed to model reflections [245] but which is not intended to model reflections in real-time.

As described in Chapter 5 polygonal geometry representations are commonly used for modeling or rendering geometry. Usually normals are defined per vertex such that smooth shading of the model can be achieved by interpolating vertex normals over rendered triangles. In the following, an approach for computing reflections for triangles with per-vertex normals by projecting vertices onto the triangle is described.

## 11.2.1 Approach

Figure 11.5(a) illustrates the setup: a triangle is defined by vertices $\mathbf{t}_1$, $\mathbf{t}_2$ and $\mathbf{t}_3$ with corresponding normals $\mathbf{n}_1$, $\mathbf{n}_2$ and $\mathbf{n}_3$. The reflection point $\mathbf{p}'$ with normal $\mathbf{n}'$ of vertex $\mathbf{p}$ given viewpoint $\mathbf{v}$ fulfills the law of perfect reflection (cf. Equation 11.2). Computation of $\mathbf{p}'$ can be done as follows: $\mathbf{p}'$ and $\mathbf{p}$ lie on a straight line with direction $\mathbf{d}' = \mathbf{p} - \mathbf{p}'$ which leads to the following equation

$$\mathbf{p} \;=\; \mathbf{p}' + k \cdot \mathbf{d}', \tag{11.8}$$

where $k \in \mathbb{R}^+$. In the following, the goal is to arrange this formula in such a way that a solution for $\mathbf{p}'$ can be found in an efficient way.

With respect to the triangle, $\mathbf{p}'$ can be defined by barycentric coordinates $\alpha$ and $\beta$

($\alpha, \beta, \alpha + \beta \in [0, 1]$) as follows (cf. Figure 11.5(b)):

$$\mathbf{p}' = \mathbf{t}_1 + \alpha(\mathbf{t}_2 - \mathbf{t}_1) + \beta(\mathbf{t}_3 - \mathbf{t}_1). \tag{11.9}$$

$\mathbf{d}'$ can be interpolated from the reflected view directions

$$\mathbf{d}_{\mathbf{t_i}} = -\frac{\mathbf{v} - \mathbf{t}_i}{\|\mathbf{v} - \mathbf{t}_i\|} + 2\left(\frac{\mathbf{v} - \mathbf{t}_i}{\|\mathbf{v} - \mathbf{t}_i\|}\mathbf{n}_i\right)\mathbf{n}_i \tag{11.10}$$

at the triangle's vertices $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3$ and can thus also be specified by barycentric coordinates (cf. Figure 11.5(c))

$$\mathbf{d}' \approx \mathbf{d}_{\mathbf{t}_1} + \alpha(\mathbf{d}_{\mathbf{t}_2} - \mathbf{d}_{\mathbf{t}_1}) + \beta(\mathbf{d}_{\mathbf{t}_3} - \mathbf{d}_{\mathbf{t}_1}). \tag{11.11}$$

Please note that correct interpolation of reflected directions requires interpolation of spherical coordinate representations, which requires introduction of undesired trigonometric functions. Fortunately, for triangles where the variation of vertex normals is small, interpolation of the directional representation produces very accurate approximations.

Putting these facts together leads to the following approximate equation system consisting of three equations and three unknowns ($\mathbf{d}_{\mathbf{t}_1}, \mathbf{d}_{\mathbf{t}_2}, \mathbf{d}_{\mathbf{t}_3}$ can be precomputed for a triangle and the current view direction):

$$\begin{aligned} \mathbf{p} &= \mathbf{p}' + k \cdot \mathbf{d}' \\ &\approx (1 - \alpha - \beta)\mathbf{t}_1 + \alpha\mathbf{t}_2 + \beta\mathbf{t}_3 + \\ &\quad k\left((1 - \alpha - \beta)\,\mathbf{d}_{\mathbf{t}_1} + \alpha\mathbf{d}_{\mathbf{t}_2} + \beta\mathbf{d}_{\mathbf{t}_3}\right). \end{aligned} \tag{11.12}$$

Unfortunately, this equation is not linear with respect to the unknowns. Solutions can nevertheless be found but require determining roots of cubic equations[1]. From the resulting equations it turns out to be most efficient to first determine $k$ (since the cubic equation for determining $k$ is much simpler than the ones for determining $\alpha$ and $\beta$) and then to compute $\alpha$ and $\beta$ with the knowledge of $k$.

[1] In concurrent work Hou et al. [226] also followed this approach but determined more complex formulas for projecting points.

### 11.2.2 Solution

Solutions for $k$ are determined by the roots of the following cubic equation:

$$\begin{aligned}
0 &= az^3 + bz^2 + cz + d \qquad (11.13)\\
a &= \det(\mathbf{d}_{\mathbf{t}_1}, \mathbf{d}_{\mathbf{t}_2}, \mathbf{d}_{\mathbf{t}_3})\\
b &= \det(\mathbf{t}_1 - \mathbf{p}, \mathbf{d}_{\mathbf{t}_2}, \mathbf{d}_{\mathbf{t}_3}) + \det(\mathbf{t}_2 - \mathbf{p}, \mathbf{d}_{\mathbf{t}_3}, \mathbf{d}_{\mathbf{t}_1}) + \det(\mathbf{t}_3 - \mathbf{p}, \mathbf{d}_{\mathbf{t}_1}, \mathbf{d}_{\mathbf{t}_2})\\
c &= \det(\mathbf{t}_1, \mathbf{p}, \mathbf{d}_{\mathbf{t}_2} - \mathbf{d}_{\mathbf{t}_3}) + \det(\mathbf{t}_2, \mathbf{p}, \mathbf{d}_{\mathbf{t}_3} - \mathbf{d}_{\mathbf{t}_1}) + \det(\mathbf{t}_3, \mathbf{p}, \mathbf{d}_{\mathbf{t}_1} - \mathbf{d}_{\mathbf{t}_2})\\
&\quad + \det(\mathbf{t}_1, \mathbf{t}_2, \mathbf{d}_{\mathbf{t}_3}) + \det(\mathbf{t}_1, \mathbf{t}_3, \mathbf{d}_{\mathbf{t}_2}) + \det(\mathbf{t}_2, \mathbf{t}_3, \mathbf{d}_{\mathbf{t}_1})\\
d &= \det(\mathbf{p}, \mathbf{t}_3, \mathbf{t}_2) + \det(\mathbf{p}, \mathbf{t}_2, \mathbf{t}_1) + \det(\mathbf{p}, \mathbf{t}_1, \mathbf{t}_3) + \det(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)
\end{aligned}$$

with

$$\det(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \det\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}$$

The roots $k_1$, $k_2$, $k_3$ can be computed applying Cardano's method [49].

$$\begin{aligned}
r &= \frac{b}{a}, \quad s = \frac{c}{a}, \quad t = \frac{d}{a}\\
p &= s - \frac{r^2}{3}, \quad q = \frac{2r^3}{27} - \frac{rs}{2} + t\\
D &= \frac{p^3}{27} + \frac{q^2}{4}\\
\rho &= \sqrt{-\frac{q^3}{27}}, \quad \phi = \frac{1}{3}\operatorname{acos}\left(-\frac{q}{2\rho}\right)\\
k_1 &= \begin{cases} \sqrt[3]{-\frac{q}{2} + \sqrt{D}} + \sqrt[3]{-\frac{q}{2} - \sqrt{D}} - \frac{r}{3} & D \geq 0 \\ 2\sqrt[3]{\rho} * \cos\phi - \frac{r}{3} & D < 0 \end{cases}
\end{aligned}$$

In the cases where $D > 0$, only one real-valued solution exists. If $D = 0$ one additional real-valued solution

$$k_2 = -\sqrt[3]{-\frac{q}{2} + \sqrt{D}} - \frac{r}{3}$$

exists if $k_1 \neq 0$. For $D < 0$ two more real-value solutions exist which are computed as follows:

$$\begin{aligned}
k_2 &= 2\sqrt[3]{\rho} * \cos\left(\phi + \frac{2\pi}{3}\right) - \frac{r}{3}\\
k_3 &= 2\sqrt[3]{\rho} * \cos\left(\phi + \frac{4\pi}{3}\right) - \frac{r}{3}
\end{aligned}$$

Having determined $k$, solutions for $\alpha$ and $\beta$ are derived from the following equations (with $(\mathbf{x})_z$ as an abbreviation for $(0,0,1)\cdot\mathbf{x}$, $(\mathbf{x})_x$ and $(\mathbf{x})_y$ analogously):

$$\begin{aligned}
N_2 &= (\mathbf{d}_{\mathbf{t}_2} \times \mathbf{d}_{\mathbf{t}_3})_z \\
N_1 &= (\mathbf{p} \times (\mathbf{d}_{\mathbf{t}_2} - \mathbf{d}_{\mathbf{t}_3} - \mathbf{t}_3))_z + (\mathbf{t}_2 \times \mathbf{t}_3)_z \\
N_0 &= (\mathbf{p} \times \mathbf{t}_2)_z \\
D_2 &= (\mathbf{d}_{\mathbf{t}_1} \times \mathbf{d}_{\mathbf{t}_2} + \mathbf{d}_{\mathbf{t}_3} \times \mathbf{d}_{\mathbf{t}_1} + \mathbf{d}_{\mathbf{t}_2} \times \mathbf{d}_{\mathbf{t}_3})_z \\
D_1 &= (\mathbf{t}_1 \times (\mathbf{d}_{\mathbf{t}_2} - \mathbf{d}_{\mathbf{t}_3}) + \mathbf{t}_2 \times (\mathbf{d}_{\mathbf{t}_3} - \mathbf{d}_{\mathbf{t}_1}) + \mathbf{t}_3 \times (\mathbf{d}_{\mathbf{t}_1} - \mathbf{d}_{\mathbf{t}_2}))_z \\
D_0 &= (\mathbf{t}_1 \times \mathbf{t}_2 + \mathbf{t}_3 \times \mathbf{t}_1 + \mathbf{t}_2 \times \mathbf{t}_3)_z \\
\alpha_i &= \frac{k_i^2 N_2 + k_i N_1 + N_0}{k_i^2 D_2 + k_i D_1 + D_0} \\
\beta_i &= \frac{(\mathbf{p} - \alpha_i (\mathbf{t}_2 - \mathbf{t}_1 + k_i \mathbf{d}_{\mathbf{t}_2} - k_i \mathbf{d}_{\mathbf{t}_1}) - \mathbf{t}_1 - k_i \mathbf{d}_{\mathbf{t}_1})_z}{(\mathbf{t}_3 - \mathbf{t}_1 + k_i \mathbf{d}_{\mathbf{t}_3} - k_i \mathbf{d}_{\mathbf{t}_1})_z}.
\end{aligned}$$

### 11.2.3 Optimization

Computation of $\alpha_i$ and $\beta_i$ according to the above equations can be implemented quite efficiently using vector arithmetic, which is available in hardware on existing graphics boards. Nevertheless the total amount of required computations can be reduced by performing computations with respect to a local coordinate system with origin $\mathbf{t}_1$, tangent $\mathbf{t}_l = \mathbf{t}_2 - \mathbf{t}_1$, normal

$$\mathbf{n}_l = \frac{\mathbf{t}_l \times (\mathbf{t}_3 - \mathbf{t}_1)}{\|\mathbf{t}_l \times (\mathbf{t}_3 - \mathbf{t}_1)\|}$$

and bitangent

$$\mathbf{b}_l = \frac{\det(\mathbf{t}_3 - \mathbf{t}_1, \mathbf{n}_l, \mathbf{t}_l)\,(\mathbf{n}_l \times \mathbf{t}_l)}{\|\mathbf{n}_l \times \mathbf{t}_l\|^2}$$

since the coordinates of the triangle's vertices reduce to

$$\tilde{\mathbf{t}}_1 = (0,0,0)^t, \tilde{\mathbf{t}}_2 = (1,0,0)^t, \tilde{\mathbf{t}}_3 = ((\mathbf{t}_3)_x, 1, 0)^t.$$

Note that in the following, entities specified w.r.t. the local coordinate system will be denoted by, e.g., $\tilde{\mathbf{t}}_1$ instead of $\mathbf{t}_1$.

**Procedure 3** renderSmoothTriangleReflections

render the scene without reflectors
**for all** reflectors $r$ **do**
  clear stencil buffer
  render $r$ into stencil buffer
  precompute subterms
  activate reflector vertex program
  send subterm results as local vertex program parameters
  render scene geometry
  deactivate reflector vertex program
**end for**

The resulting simplified cubic equations for determining $k_1$, $k_2$ and $k_3$ are:

$$\begin{aligned} 0 &= \tilde{a}z^3 + \tilde{b}z^2 + \tilde{c}z + \tilde{d} \\ \tilde{a} &= \det\left(\tilde{\mathbf{d}}_{\mathbf{t}_1}, \tilde{\mathbf{d}}_{\mathbf{t}_2}, \tilde{\mathbf{d}}_{\mathbf{t}_3}\right) \\ \tilde{b} &= \det\left(-\tilde{\mathbf{p}}, \tilde{\mathbf{d}}_{\mathbf{t}_2}, \tilde{\mathbf{d}}_{\mathbf{t}_3}\right) + \det\left(\tilde{\mathbf{t}}_2 - \tilde{\mathbf{p}}, \tilde{\mathbf{d}}_{\mathbf{t}_3}, \tilde{\mathbf{d}}_{\mathbf{t}_1}\right) + \det\left(\tilde{\mathbf{t}}_3 - \tilde{\mathbf{p}}, \tilde{\mathbf{d}}_{\mathbf{t}_1}, \tilde{\mathbf{d}}_{\mathbf{t}_2}\right) \\ \tilde{c} &= \left(\tilde{\mathbf{p}} \times \left(\tilde{\mathbf{d}}_{\mathbf{t}_3} - \tilde{\mathbf{d}}_{\mathbf{t}_1}\right)\right)_x + \det\left(\tilde{\mathbf{t}}_3, \tilde{\mathbf{p}}, \tilde{\mathbf{d}}_{\mathbf{t}_1} - \tilde{\mathbf{d}}_{\mathbf{t}_2}\right) + \left(\tilde{\mathbf{d}}_{\mathbf{t}_1}\right)_z \\ \tilde{d} &= -\left(\tilde{\mathbf{p}}\right)_x . \end{aligned} \tag{11.14}$$

For $\alpha_i$ and $\beta_i$ the simplified equations are:

$$\begin{aligned} D_2 &= \left(\tilde{\mathbf{d}}_{\mathbf{t}_1} \times \tilde{\mathbf{d}}_{\mathbf{t}_2} + \tilde{\mathbf{d}}_{\mathbf{t}_3} \times \tilde{\mathbf{d}}_{\mathbf{t}_1} + \tilde{\mathbf{d}}_{\mathbf{t}_2} \times \tilde{\mathbf{d}}_{\mathbf{t}_3}\right)_z \\ \alpha_i &= \frac{k_i^2 \left(\tilde{\mathbf{d}}_{\mathbf{t}_2} \times \tilde{\mathbf{d}}_{\mathbf{t}_3}\right)_z + k_i \left(\tilde{\mathbf{p}} \times \left(\tilde{\mathbf{d}}_{\mathbf{t}_2} - \tilde{\mathbf{d}}_{\mathbf{t}_3} - \mathbf{t}_3\right)\right)_z + k_i - \left(\tilde{\mathbf{p}}\right)_y}{k_i^2 D_2 + k_i \left(\left(\tilde{\mathbf{d}}_{\mathbf{t}_3} - \tilde{\mathbf{d}}_{\mathbf{t}_1}\right)_y + \left(\mathbf{t}_3 \times \left(\tilde{\mathbf{d}}_{\mathbf{t}_1} - \tilde{\mathbf{d}}_{\mathbf{t}_2}\right)\right)_z\right) + 1} \\ \beta_i &= \frac{\left(\tilde{\mathbf{p}} - \alpha_i \left(k_i \tilde{\mathbf{d}}_{\mathbf{t}_2} - k_i \tilde{\mathbf{d}}_{\mathbf{t}_1}\right) - k_i \tilde{\mathbf{d}}_{\mathbf{t}_1}\right)_z}{\left(k_i \tilde{\mathbf{d}}_{\mathbf{t}_3} - k_i \tilde{\mathbf{d}}_{\mathbf{t}_1}\right)_z}. \end{aligned}$$

Please note that encoding directions in a two plane parameterization [305], where one plane is defined by the triangle plane and the second one by a parallel plane with distance one, results in a much simpler, unique solution for $k$. Unfortunately, interpolating reflected directions in this representation leads to arbitrarily bad results for grazing view angles w.r.t. the triangle plane.

The above equations are evaluated for a large number of vertices which are to be mirrored in the triangle. Therefore, evaluation efficiency can further be optimized by precomputing all (sub-)terms independent of $\tilde{\mathbf{p}}$, since they depend on triangle parameters and the current viewing direction only.

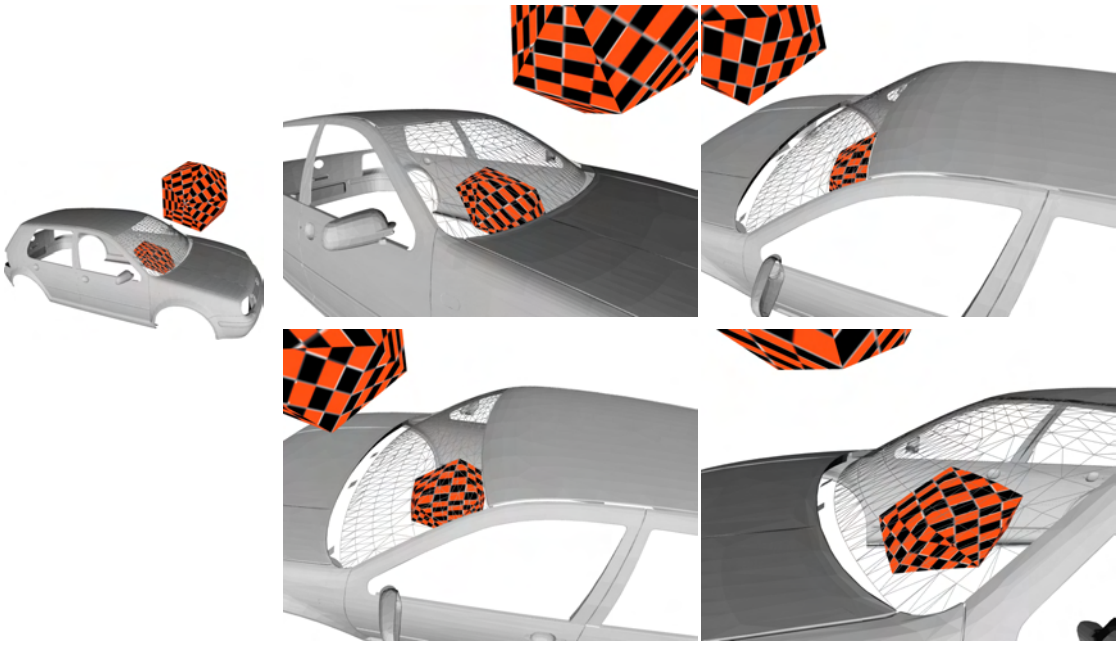


**Fig. 11.6: Reflections in windshield approximated by triangles with per-vertex normals from outside the car. Left: setup, middle and right: different views.**

### 11.2.4 Rendering Algorithm

The algorithm for rendering reflections in triangles with varying normals is essentially the same as for rendering reflections in Bézier surfaces and is illustrated by the pseudocode of Procedure 3. The reflector vertex program transforms vertices into the triangle's local coordinate system and evaluates the equations for determining reflection points. As shown above, three locations for the reflection point may exist among which the one closest to the center of the triangle is chosen. Again, reflections have to be depth-corrected to guarantee correct occlusion of reflected objects (see Section 11.1.4).

Figure 11.6 shows screenshots from the implemented method for the case of a convex reflector, Figure 11.7 shows results for reflections in a concave reflector (note that only the textured object is supposed to be mirrored in the windshield). The right bottom image in Figure 11.7 illustrates that the algorithm can handle cases where inflexion points or inflexion lines exist.

### 11.2.5 Discussion of Results

Compared to the reflection algorithm for Bézier surfaces, the approach based on triangles with per-vertex normals gives much more accurate results. For the case of convex surfaces correct reflection points are always found. The approach is even capable of

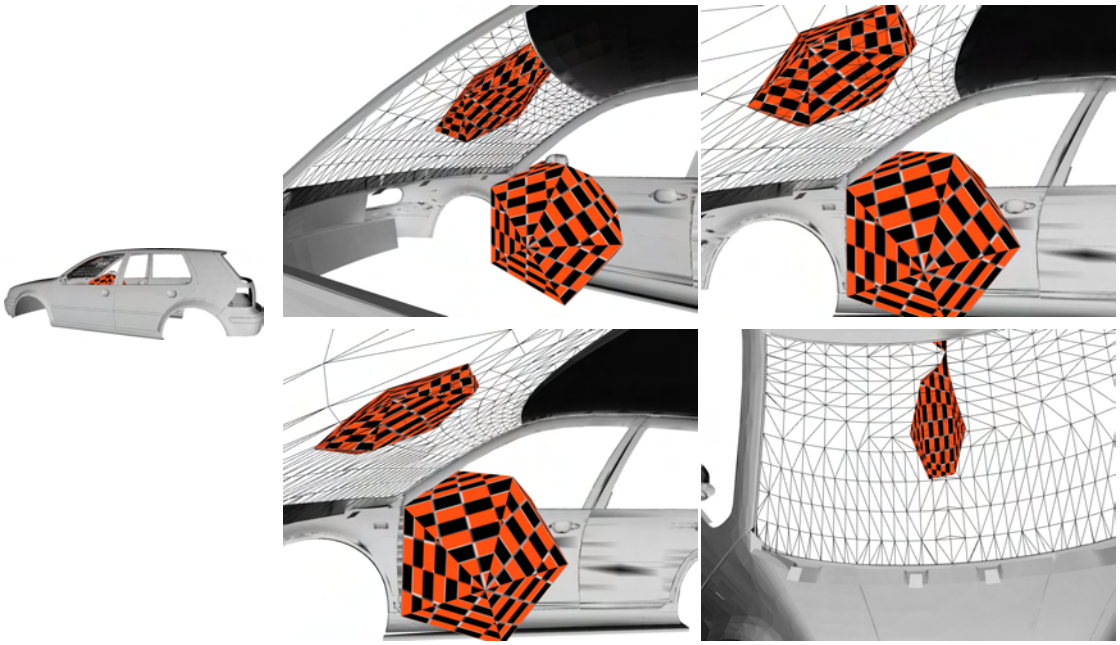


**Fig. 11.7: Reflections in windshield approximated by triangles with per-vertex normals from inside the car. Left: setup, middle and right: different views.**

handling concave surfaces to some extent (cf. Figure 11.7), since the approximation of a curved surface by triangles and the approximate interpolation of reflection directions prohibits cases where an infinite number of projection points exist for a single triangle. Although this leads to a much more simple and robust algorithm, it naturally limits the degree of realism achievable with this approach.

Unfortunately, the approach cannot be combined with rendering triangles in a correct way due to two reasons. First, lines connecting vertices will be rendered as straight lines. Yet, they should appear curved due to the curved reflector surface. While this problem might be acceptable, the second problem is clearly not. As mentioned above, solving the equations for computing reflections leads to three possible locations for the

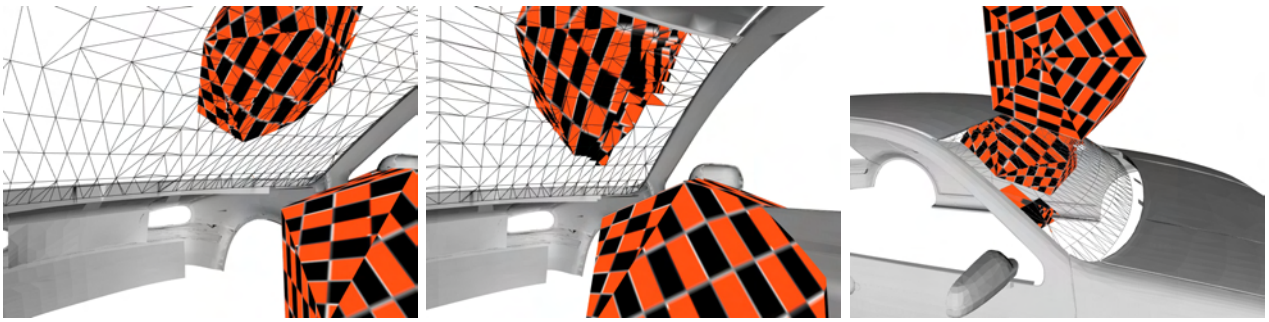


**Fig. 11.8: Problems when computing reflections in windshield approximated by triangles with per-vertex normals due to triangle assembly.**

**Fig. 11.9: Problems like false reflections vanish when rendering point primitives.**

projected point. While tests suggest that this causes no problem for vertices projecting to the triangle, projection points outside the triangle need to be considered as well. Despite not being rendered themselves, these positions influence the triangles being assembled in the graphics pipeline. Therefore, such problematic projected vertices cannot simply be discarded but an as accurate as possible location has to be chosen even if all three vertices of a triangle do not project to the reflector triangle. Unfortunately, without further knowledge the best possible location among the three possible ones cannot be determined. Figure 11.8 shows results of this problem for convex and concave reflectors: While the left image shows jaggy edges only, the middle image features clearly visible errors. In the right image some vertices of the reflected icosahedron are positioned below the plane of some reflector triangles. Figure 11.9 illustrates that such problems can be omitted when combining the approach with point-based rendering (the right image shows no projected points as expected). Another solution to this problem was proposed in concurrent work by Hou et al. [226]. Instead of computing projection points in the vertex shader, the authors propose to compute projection points in the fragment shader, which implements a simple ray-caster intersecting the reflected ray with potentially visible triangles. The vertex shader simply serves to restrict the projection area as much as possible. This approach additionally solves the problem that straight lines project to straight lines. Yet, it requires far more computational power than the proposed solution.

Another problem of the approach is the rendering performance since the whole scene has to be rendered for every reflecting triangle and since usually many triangles are required to approximate a curved surface in an accurate manner. Optimizations to this problem can again be obtained using spatial subdivision structures. When rendering reflections of a specific triangle reflector, all points that project to the surface lie within the subspace spanned by the triangle's vertices and the reflected view directions. Nevertheless, efficient combinations of the approach with large models seems to be very difficult.

# 11.3 GPU Raycasting

The previous sections showed that forward-mapping approaches for computing reflections in curved surfaces have limited success only and – in fact – can have limited success only. Therefore, this section presents an inverse-mapping approach based on raytracing.

## 11.3.1 Approach

As mentioned in Section 8.2.3, recent advances in raytracing algorithms have improved the rendering speed of raytracing steadily and might eventually lead to widespread use of this technology in real-time applications. Yet, current approaches are still not capable of rendering high resolution images at interactive frame rates on a single commodity PC. Additionally, integration with existing applications based on rasterization, which still dominate real-time rendering software, is difficult.

To overcome this problem and to alleviate the computational power of GPUs, raytracing was ported onto the GPU, where a ray is traced for each rasterized fragment. Unfortunately, the success of these methods is very limited, which stems from two major problems.

First, intersecting rays and triangles requires a rather complex test. In combination with large numbers of triangles, this usually overextends even the massively parallel computation capabilities of GPUs. Unlike CPU implementations, GPU algorithms cannot easily amortize these costs by simultaneously tracing multiple rays.

Second, to decrease the number of ray triangle intersections, optimized spatial subdivision data structures are necessary. Due to the resources available on graphics hardware, existing GPU raytracing approaches employ regular grids which do not adapt to the geometry (the recent, parallel work of Foley and Sugerman [149] is a remarkable exception). This either leads to a large number of subdivisions and thus large numbers of unnecessarily traversed cells and high memory requirements, or an insufficient reduction of the number of intersection tests. A very nice and detailed documentation of both problems can be found in [66].

In order to solve these two problems, two principal modifications for raytracing algorithms on the GPU are suggested [356]: First, the use of BRDF enhanced volumetric representations of the geometry which can efficiently be stored in textures and which allow much simpler intersection tests. The price for this simplicity are lower resolution results for higher-order intersections (the first order intersections are accurately realized by rasterization hardware). For most applications this poses no problem as long as the locality of the reflection or shadow is correct – which is guaranteed by the presented

**Fig. 11.10: Windshield of a car without (left) and with (right) interactively rendered reflections.**

approach. In addition, future graphics hardware will allow even higher resolutions, thus increasing the achieved rendering quality. Therefore, this tradeoff seems to be well suited for a large range of applications.

As a second modification, computing this volumetric representation by recursively subdividing the scene geometry with an octree is proposed. The octree representation is stored in texture memory and raytraced for rendering, which has never been done on the GPU before. Due to limitations of existing GPUs, the level of subdivision is bound to a fixed depth (currently 9-10) but experiments showed that even at this fixed level very good results can be achieved (cf. Figure 11.10). Details on both modifications are given in the following.

### Volumetric Representation

Raytracing volumetric representations has been shown to be very efficient on the CPU [404]. Mapped to graphics hardware which supports trilinear interpolation, the approach simply requires continuous sampling from the volumetric dataset along the ray direction until an opaque value is found or until the accumulated opacity exceeds a given threshold.

Since scenes employed for real-time rendering usually consist of triangles, a volumetric representations has to be derived first. Given some spatial subdivision scheme, the scene is partitioned into cells. For each cell a (multi-dimensional) value is stored, where the stored components depend on the available material properties. The results shown here store an average normal and a material per cell. To compute the average normal, for each cell $c_i$ and triangle $\mathbf{t}_j$ contained in the cell the area $a_{i,j}$ of the intersection of $c_i$ and $\mathbf{t}_j$ is computed. The average normal $\mathbf{n}_{c_i}$ is then computed as $\mathbf{n}_{c_i} = \sum a_{i,j}\mathbf{n}_j$ where $\mathbf{n}_j$ denotes the normal of triangle $\mathbf{t}_j$. Computing the cell's material is detailed in section 11.3.2.

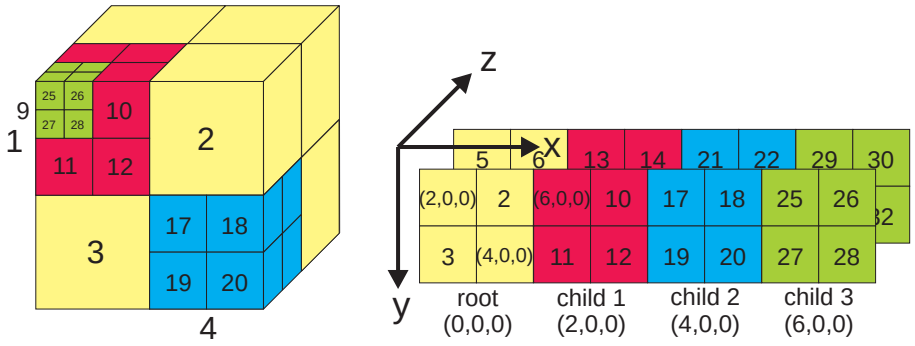



**Fig. 11.11: Mapping of octree (left) to texture memory (right).**

## Octree Subdivision

As mentioned above, current GPU raytracing algorithms employ grids for spatial subdivision since the traversal algorithm of Amanatides and Woo [5] easily maps to the capabilities of fragment programs. Efficient implementation of more complex but simultaneously more efficient, hierarchical strategies on the GPU is difficult due to the limited amount of temporary storage necessary for efficient recursive or iterative traversal.

Nevertheless, the proposed solution employs spatial subdivision by an octree since it offers a good compromise between adaptability (which a regular grid is missing) and subdivision depth (which is usually much higher using a kd-tree).

While hierarchical data structures are typically implemented using references or pointers into main memory on the CPU, fragment programs do not directly support this concept. Instead, dependent texture lookups replace dereferencing of pointers. Figure 11.11 gives an overview of the mapping from the hierarchical octree structure (left) to a 3D texture (right), which is nearly identical to the layout in [298]. The root of the octree is located at texture coordinates $(0,0,0)-(1,1,1)$. It stores pointers to child nodes that represent inner nodes of the octree hierarchy and the properties of nodes that represent leaf nodes. In the example given, the root's children are named $1\ldots8$ with nodes $1$ and $4$ being inner nodes and the remaining ones being leaf nodes. The inner nodes $1$ and $4$ are stored at texture coordinates $(2,0,0)-(3,1,1)$ and $(4,0,0)-(5,1,1)$ respectively. Inner node children themselves hold references to their respective inner node children and the properties of their leaf node children.

The implementation of the proposed method stores references as three bytes, one byte for each texture dimension, allowing a total octree texture size of $256^3$ which suffices for octrees of depth nine to ten according to our experiments. For leaf nodes the average normal and material properties are stored instead of a texture coordinate. If the leaf contains no geometry, the material property is set to a reserved value (e.g., 0). To

**Fig. 11.12: Octree Representation of the Mercedes C class interior. Top left: complete car, right: interior. Bottom: two views of octree representation of car interior and view of the steering wheel.**

distinguish references to inner nodes from leaf cases, the material property is set to a second reserved value (e.g., 1) whenever a reference is encoded.

Storage requirements for the octree are rather moderate: For the case of the Mercedes shown in Figure 11.12, the octree texture requires 32 MB at depth nine (maximum $512^3$ cells), corresponding to a minimum cell edge length of about 0.7 cm, and 128 MB at depth ten. Encoding the same resolutions with a grid would require 512 MB and 4 GB. Increasing the resolution by a factor of two increases the memory requirements of an octree by an approximate factor of four while the regular grid features a factor of eight. Figure 11.12 gives an impression of the approximation quality achieved by an octree of depth eight.

The hierarchical structure of the octree enables a simple, yet efficient way to incorporate regions of interest: The user can define a set of especially interesting reflectors in the geometry and then the necessary resolution of a part of the geometry can be estimated by the minimal distance of the part to the set of reflectors.

The pseudocode of the traversal algorithm is listed in Procedure 4. The initial part checks whether the ray intersects the octree. If the ray misses the octree, the scene's background color is returned. Otherwise, an initial traversal point is computed that either represents the ray origin if it is located inside the octree or the first intersection of the ray and the octree's bounding box. Additionally, the stack used during traversal is cleared and the octree root node is stored as the cell where traversal proceeds.

In the following traversal loop the location of the traversal point is propagated along the ray direction until it is located either within a leaf cell that contains geometry or outside the root node. Each iteration first performs a containment test: It determines

**Procedure 4** Traverse( ray, octree )

```
if Hit( ray, octree ) then
    point ← ComputeInitialPoint( ray, octree )
else
    return background_color
end if
father ← Root( octree )
stack ← empty_stack
loop
    cell ← ComputeSubcell( point, father )
    if InnerNode( cell) then
        Push( stack, father )
        father ← cell
    else
        if HasGeometry( cell ) then
            return Shade( cell )
        end if
        point ← ComputeExitPoint( ray, cell ) + ε· Dir( ray )
        while Outside( point, father ) do
            if Empty( stack ) then
                return background_color;
            end if
            cell ← father
            father ← Pop( stack )
        end while
    end if
end loop
```

in which subcell of the current cell the traversal point is located. If this subcell is an inner node of the octree hierarchy, the current cell is pushed on the stack and traversal continues with the respective subcell. Otherwise, the algorithm checks whether the subcell contains geometry in which case it is shaded and the resulting color is returned. Otherwise the new location of the traversal point is computed as the position where the ray leaves the subcell. To guarantee a robust containment test, the traversal point is slightly advanced along the ray direction. For the updated traversal point the algorithm iteratively checks if it is located outside the current cell, in which case traversal has to continue with the father of the current cell, which is popped from the stack. If no such father exists on the stack, the traversal point is located outside the root node and traversal ends by returning the background color.

All parts but the stack implementation easily map to fragment programs [393]. Implementing a stack is difficult [130] since fragment programs currently prohibit dynamic

indexing of arrays. Therefore, the method implements access to the stack by explicitly encoding the case for each octree level and selecting the matching case during traversal by nested IF . . . ELSEIF constructs.

Since the number of temporary registers of a fragment program is limited, the maximal traversal depth of the octree is limited on the GPU. For recent graphics boards like the nVIDIA GeForce 7800 the GPU offers 32 registers. The traversal method requires two three-component registers per octree recursion level (one storing the texture coordinates of the current father, a second one storing the center point of the cell), allowing to encode ten octree levels for a maximum number of $1024^3$ cells. Future versions of GPUs are very likely to feature more registers and will therefore allow octrees of even higher resolution, but as Figure 11.10 shows, even a depth of nine leads to reasonably accurate images.

A very nice feature of employing hierarchical, volumetric data structures while computing reflections off curved surfaces is that they allow for efficient anti-aliasing. Since rays corresponding to pixels need to be interpreted as ray cones instead of lines without extent, anti-aliasing requires averaging the resulting colors over the area of intersection of ray-cones and scene geometry. Similar to tracing dilating rays against multiresolution point clouds as done by Wand and Straßer [551], raytracing hierarchical, volumetric data sets allows for efficient anti-aliasing. Assuming a locally flat geometry of the ray origin (in the given case the intersection of a first-order eye ray and the scene geometry), the width of a reflected ray increases linearly with the traveled distance. Tracking this distance and restricting the maximum recursion depth (depending on the width of the ray cone) during traversal results in anti-aliased rendering results.

### 11.3.2 Material Assignment

Maybe as important for rendering realistic reflections as accurately representing the geometry are precise material properties due to their direct impact on object appearance. When converting surface models into the octree representation, the surface materials have to be stored per octree cell in an as accurate as possible manner. Depending on the type of materials supplied and the expected rendering accuracy, two approaches directly apply: One can either select the material dominating the surface area contained in a cell or one can fit a specific bidirectional reflectance function (BRDF) per octree cell.

Realizing the first approach can be done in a straightforward manner (Gobbetti and Marton [163] used a very similar method in a recent publication). Analogous to computing average normals, one first determines the surface area $a_{i,j}$ of all triangles $\mathbf{t}_j$ contained in cell $c_i$ intersection. Then the material $m_{c_i}$ to be stored for the cell $c_i$ is chosen as the one where the sum of areas $a_{i,j}$ of triangles with matching assigned material $m_{c_i}$

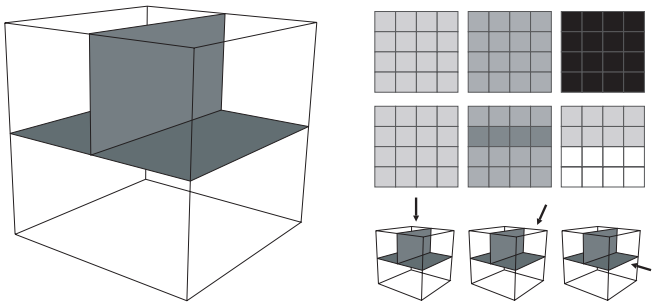


**Fig. 11.13: Errors introduced by choosing a single material per cell. Left: cell geometry with uniform, diffuse material. Right: views of the cell for different view directions and a 4×4 viewport. Top: single BRDF approximation, bottom: original appearance.**

is maximized. Storing the material per octree cell can be realized by a single material index, which nicely fits into the fourth component of the vector stored with each cell. Taking into account two reserved material indices for empty cells and those containing pointers to children, this approach allows to encode up to 254 materials for a scene - a number which suffices for most scenarios.

Unfortunately, as visualized by Figure 11.13, many visual effects cannot be captured by choosing a single material or averaging over existing contained materials per cell. If, e.g., the cell contains a T-shaped geometry, the average normal will equal the normal of the T base. A view of the original geometry from top (leftmost grid) will yield a uniform gray area, which equals the appearance when using a single material. When viewing the material from a lower angle, the upright part becomes visible, and the resulting color will be different from the one on the base. As a result, the average color value does not match the one from the single material any more. Another problem occurs when viewing the geometry from a direction parallel to the base: Now, not only the average color differs from the one computed using the stored material but additionally half of the pixels remain transparent.

Both these effects can be taken into account by fitting an extended, average BRDF per octree cell which additionally encodes a transparency value. Before the average BRDF can be computed, an average local coordinate system has to be defined for the cell, which requires the computation of an average tangent in addition to the already computed average normal. Tangents for each of the triangles can be computed either from existing texture coordinates or applying a simple set of rules. A standard example for such rules is to compute the tangent as the cross product of the global $y$ axis and the normal direction as long as the angle between the $y$ axis and the normal is large enough. Otherwise the $x$ axis is taken instead of the $y$ axis. Averaging the tangents is done analogously as for the normals.

Next, a set of view- and light directions is defined at which the cell's average BRDF is

sampled. The sample directions need to be taken from the full sphere of directions since the geometry located in the cell needs not be flat. For high-quality BRDF reconstruction, a rather dense sampling of 98 view- and light-directions is employed. The 98 directions correspond to the pixels of a front- and a backfacing parabolic map with resolution $7\times7$ and are therefore nearly equally distributed over the sphere of directions.

Now, for each cell and each pair of view and light directions the cell is rendered and the resulting colors and transparency values are averaged. The $98^2$ average values represent the per-cell average BRDF samples and are stored in a vector. Note, that these samples include shadows from surrounding geometry if not only a local illumination model is evaluated.

Since the resulting data set is huge (for the example in Figure 11.12 which contains about two million filled leaf cells, about 77 GBs would be required to store the sampled BRDFs), the data needs to be compressed and compression should occur on-the-fly in order to avoid writing such a huge amount of data to disk. The proposed method proposes application of local principal component analysis (PCA) compression (a combination of clustering and PCA) to the data, which has been employed for compression of a large number of BRDFs in the context of bidirectional texture functions rendering already [369]. To avoid computing the whole data at once, the method first selects a subset of the filled cells, then computes their BRDFs and compresses them using local PCA. The remaining BRDFs then only need to be fit into the correct cluster. Standard recommendations suggest to use 5-10% of the data for training but tests determined that about 10000 samples yield good results already.

Sampling the BRDFs for about two million cells requires a very long time. Fortunately, the octree subdivision yields many cells where the contained geometry carries a single material only and is nearly planar (i.e. normal deviation smaller than five degrees). For all these cells (about 1.3 million in the example) one can sample and fit the BRDF of the respective material once and reuse the data for all corresponding cells.

Unlike the first method for material encoding which requires a normal and a material index per cell only, the BRDF method needs to store normals, tangents and BRDF reconstruction parameters (a cluster index and four PCA weights per cell). Therefore the encoding of leaf octree cells is changed such that they store a texture coordinate into a material texture and the cluster index as fourth component. As for the material index, reserved values for the cluster index mark empty nodes and references to children. While normal and tangent values are efficiently stored in a byte-valued RGB texture, the four weights are stored as 16 bit float values. Storing Eigen-BRDFs and reconstructing per-cell BRDFs follows the scheme suggested by Müller et al. [369].

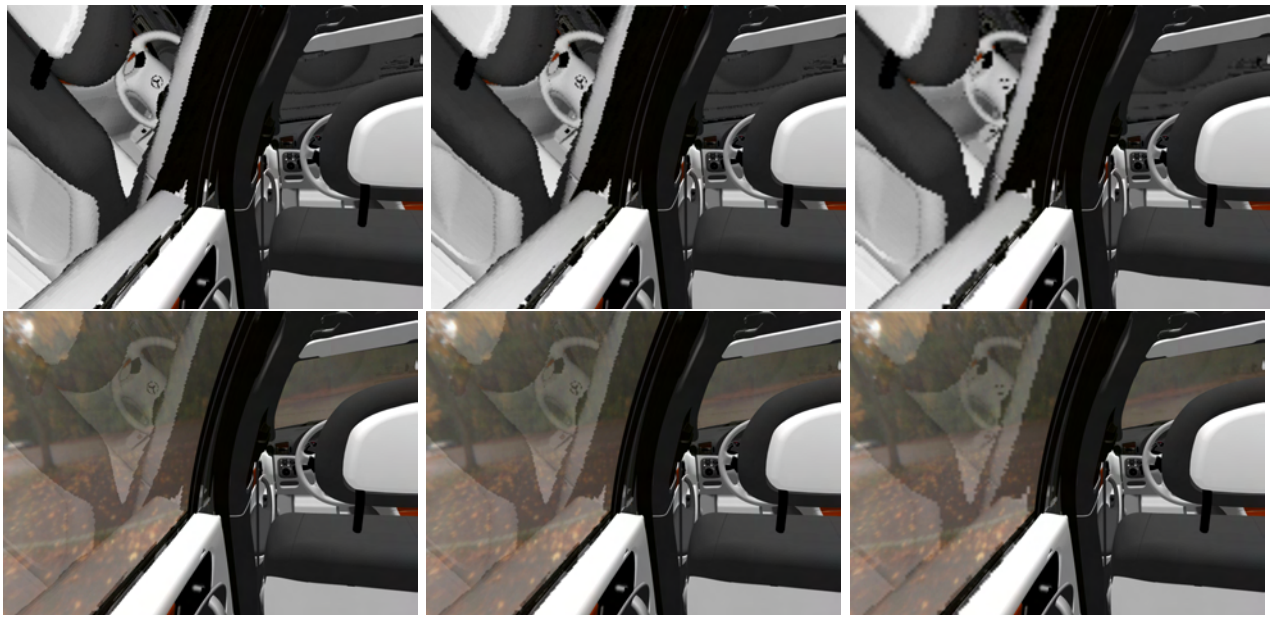


**Fig. 11.14: Comparison of rendering resolution. View from back seat. Left: full, middle: half, right: quarter resolution. Top: total reflection, bottom: with transparency.**

### 11.3.3 Optimizations

In order to optimize fragment tracing with respect to rendering time, one has to note first that the rendering speed of the algorithm is directly limited by the number of fragments handed to the fragment program. As one can see in Figure 11.14 the individual octree cells (which appear as regions of constant brightness) are quite visible. An approach for interpolation of these colors that additionally reduces the amount of fragments is therefore to render the image into a lower-resolution texture which is later used to cover the reflecting parts while enabling bilinear texture filtering. This approach, which follows a similar idea than the vertex tracing method [530], makes the frame rate independent of the display resolution which turns out to be a very nice feature for high-resolution displays like power-walls and caves. As the images in Figure 11.14 show, the reflections appear to be a bit blurred or frayed if the resolution is reduced. Yet, this effect becomes nearly invisible if a partially transparent reflector is used.

Another optimization especially suitable for reflections in transparent objects is based on the Fresnel term, which quantifies how much of the incoming light is transmitted and how much is reflected. If the amount of reflected light is much lower than the amount of transmitted light, one can simply neglect the contribution from the reflection, i.e., the reflected ray needs not be traced. The amount of transmitted light can often easily be determined (in the images in Figure 11.15 this requires a single cube texture lookup only). In the conducted tests it was assumed that the reflection from a surface can

**Fig. 11.15: Optimization by Avoidance. Left: no reflections, right: with reflections. The differences are invisible against the bright background.**

at most be as intense as the light sources in the scene. If the product of maximum incoming intensity and Fresnel term is at most 5% of the brightness of the transmitted light or if the transmitted light is already saturated, the reflected ray needs not be traced. Computation of the fraction $r$ of incoming light reflected due to the Fresnel equations can efficiently be done using Schlick's approximation [453]

$$r = r_0 + (1 - r_0)(1 - \cos\theta_i)^5, \tag{11.15}$$

where $r_0$ denotes the material dependent Fresnel reflection coefficient and $\theta_i$ the incident angle. Figure 11.15 illustrates the effect: In the part where the bright pixels of the surrounding environment map are visible, the reflections do not significantly contribute to the final image. Therefore, the rays at these pixels can be omitted, reducing overall rendering time by about 30%.

### 11.3.4 Results

Time requirements for constructing the octree and storing it in a 3D texture range from 4 minutes for an octree of depth 7 to 15 minutes at depth 9 on a Pentium 4 2.4 GHz processor with 1 GB main memory if the simple method for material selection is chosen. Computing exact BRDFs requires much longer – about 25 hours – but computation can easily be parallelized over multiple computers once the local PCA of the initial 10000 samples is computed.

As mentioned already, the memory requirements for the octree texture are rather small: 2 MB at depth 7 and increasing by an approximate factor of four for every further level. Of course, the exact factor depends on the given scene, but a factor of four comes hardly at a surprise considering that a surface representation is voxelized.

**Fig. 11.16: Reflections computed with standard raytracer.**

| octree depth | $160 \times 120$ | $320 \times 240$ | $640 \times 480$ |
|---|---|---|---|
| 7 | 7.8 fps | 4.0 fps | 1.7 fps |
| 8 | 6.1 fps | 2.7 fps | 1.1 fps |
| 9 | 4.5 fps | 1.8 fps | 0.7 fps |

**Tab. 11.1: Rendering performance on a GeForce 6800 Ultra**

Encoding simple materials requires 4 kB only, the per-cell BRDF data sums up to about 40 MB.

The rendering quality and accuracy achieved with the fragment tracing approach fulfills the requirements of many applications. Compared to images computed with standard raytracing algorithms like Figure 11.16 (computed with the Maya raytracer in many seconds on a single PC) reflections computed by the proposed approach appear blocky due to the limited resolution of the octree. Nevertheless, the tradeoff between speed and resolution turns out to be very useful for many applications. As Figure 11.17 shows, the approach can for example be usefully employed in Virtual Prototyping.

The run-time performance of fragment tracing mainly depends on the maximum depth of the octree and the amount of fragments traced (cache locality, which improves at higher resolutions, plays a factor as well). As Table 11.3.4 shows, rendering times for a moderate octree resolution of level seven (which suffices for small objects) and a reasonable rendering resolution are highly interactive. For higher resolutions of the octree which are required for large, detailed models as those used in Virtual Prototyping, which commonly contain more than two million triangles, frame rates are still interactive. Note that the frame-rates are computed for the car model consisting of about two million polygons.

An efficient way for run-time optimization of the fragment tracing approach is parallelization of the approach using emerging technologies like nVIDIA's Scalable Link Interface, which enables parallel use of multiple graphics boards in standard PCs. An-

**Fig. 11.17: Slight material changes drastically influence the reflections. Left: original, middle: slight specular factor added, right: slightly increased diffuse and specular reflection properties. Such cases should be detected using Virtual Prototyping.**

other option are clusters of PCs where each computer renders a small piece of a window only. Obviously, the scene and octree data has to be replicated on all participating computers in such a scenario.

Additionally, future advances in graphics hardware are very likely to improve the performance of this method both in terms of run-time and rendering quality. Upcoming graphics chips will feature more parallel pixel pipelines and higher clock frequencies, boosting the performance of fillrate-limited applications based on tracing fragments. In addition, the instruction set is very likely to be extended in the near future. This might enable a more efficient stack implementation by either introducing a multicase selection statement (SWITCH) or allowing dynamic addressing of registers. Together with an increased number of available registers, this will allow more or even an arbitrary number of octree levels, effectively increasing the resolution of the method.

### 11.3.5 Discussion

The presented approach for interactively rendering reflections by tracing rays through a volumetric, octree-based representation of the scene on the GPU produces good results for a large variety of applications. Unlike the forward-mapping approaches it features no principle limitations: it handles arbitrary reflector surfaces, large amounts of geometry, and allows for efficient anti-aliasing.

The first point is achieved since the fragment shader only requires a position and normal for every fragment it processes. Therefore every type of reflector surface which allows for determination of depth and normal per fragment can be used. This permits integration with state-of-the-art rendering methods for parametric surfaces (e.g., [184]), which are widely used in industrial prototyping, and other surface representations.

The second point is achieved by encoding geometry as filled cells of an octree which

allows for simulating reflections of large models (results for models with a complexity as typically employed in Virtual Prototyping were shown) – a clear advantage over related approaches. The drawback of this approach is loss of resolution, but this can often be accepted as shown by the results.

The third point is achieved exploiting the hierarchical, volumetric representation, which allows for efficient anti-aliasing by tracing dilating rays – a huge advantage over the standard approach for raytracing, which demands casting several rays per pixel.

As a result, the presented approach achieves very good results on standard graphics cards as long as the limited resolution of reflections is acceptable.

# Part V

# Verification of Predictive Rendering Methods

# Chapter 12

# State of the Art

The central goal of this thesis is description of techniques that contribute to the development of predictive Virtual Reality systems. In the previous parts, suitable material, light and geometry representations, and techniques for rendering very accurate results at interactive frame rates were presented. Obviously, all these efforts will only have a significant impact on industrial applications like virtual prototyping if the users trust in the accuracy of the generated images. Clearly, the most definite way to provide confidence is proving the correctness of simulated results.

The need for verification of rendering techniques was pointed out as soon as the first physically based rendering techniques became available: With the introduction of the Radiosity method in 1984, Goral et al. [168] already conducted studies concerning the correctness of their solutions. While initial results based on naive, visual comparisons of rendered images and reality, later approaches employed more elaborate techniques, leading to better founded conclusions.

An important step towards rigorous verification of predictive rendering techniques was the introduction of a verification framework by Greenberg et al. [175]. They pointed out the necessity to validate all stages related to predictive image generation (cf. Figure 12.1), i.e., scene modeling, light transport simulation, and display. Since errors in earlier stages can influence the accuracy of a later stage the authors proposed individual verification of each step combined with validation of the interplay of stages. In the following, existing methods performing such verifications are described.

## 12.1 Verification of Modeling Step

As noted in Section 1.3 already, predictive renderings cannot be generated without having accurate input to the light transport simulation algorithms. Given the standard way

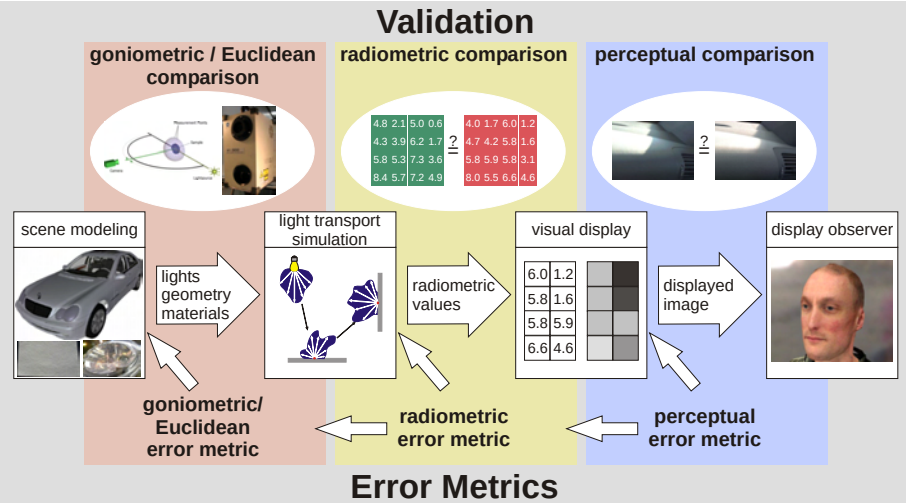



**Fig. 12.1: Validation of the predictive image generation process [175].**

of modeling scenes this requires physically accurate light source, material and geometry representations.

## Light Sources

Due to the extreme importance of accurate light source data for illumination simulations and everyday use, major light source manufacturers acquire such data routinely today. The websites of such companies often allow download of such data, e.g., goniometric diagrams, and customers may request special data from manufacturers.

Due to the widespread use in industry, measurement devices like goniophotometers can be considered accurate today. Nevertheless, for accurate modeling such data is not sufficient for three reasons. First, data is measured in photometric units only while spectral radiometric data is required. Second, typical measurements are limited to far-field data while close-range effects require detailed close-range data. Measurement techniques for such data exist [164] but are not widely used today. Third, the properties of light sources vary depending on their life-time, their temperature, electricity supply, and many other influences. Measurement and control of these influences is difficult and currently only a maintenance factor characterizing the decrease of emitted light energy over time is used. Acquisition and verification of more accurate data will therefore be necessary in the future.

## Geometry

The current use of Virtual Reality in industrial prototyping – currently probably the most prominent application area requiring predictive results – leads to a special role for geometry. Typically, new products consisting of purely virtual geometry and existing materials are simulated in environments containing real light sources. Therefore, verification of geometry is neither necessary nor possible in such a scenario. Typically, VR users rely on the abilities of CAD modeling systems and experts to create accurate geometric models.

In design workflows that iterate between digital and physical model building or that derive digital models from physical ones it is common practice to verify the geometry's accuracy by comparing real and virtual geometry. Two principle approaches exist. First, it is possible to create physical replications of virtual geometry using 3D printing technologies like stereo-lithography. These models can afterwards be compared with original models. Unfortunately, due to size limitations of 3D printing devices it is usually difficult to produce physical replications at the original scale. Additionally, the approach introduces long production times and high costs for creating real models. Therefore, the second approach takes the opposite approach and transfers the original physical model into a digital one using 3D scanning technology [462]. CAD modeling experts can use this digitized data to guide the modeling process, or the measured points can be matched and compared with existing digital models.

A severe simplification of accurate geometry modeling is introduced by neglecting small-scale geometric detail which is considered to be part of the surface materials and is thus measured or defined separately. Accepting this separation, which is common practice for specifying virtual scenes, one can say that accurate modeling of geometry is a well established process today since it has been used in CAD modeling for a long time already.

## Materials

Unlike acquisition of light source properties and modeling of geometry, acquisition and verification of materials has become an interesting area for industry just recently. Therefore, approaches are less settled and fewer verified measurement methods exist. Only for relatively simple material representations like BRDFs existing goniometer-like acquisition setups have found their way into mainstream application.

For more complex material representations that include a larger variety of effects than BRDFs various acquisition methods were described in Chapter 2. Although researchers pay a great amount of attention to accurately calibrating their measurement

setups few of the measurements were actually verified. For the case of custom BRDF measurements Schregle and Wienold [459] present a number of possible verification steps testing BRDF reciprocity and energy preservation. Nevertheless, these steps are not sufficient to verify the correctness of measurements. An extension of this approach was presented by Havran et al. [208] who compare measured BTFs with measured BRDFs by computing mean BRDFs from the BTFs.

Due to the enormous complexity of materials verification methods for material acquisition setups are not established yet. While BRDF acquisition methods seem to be sufficiently accurate, such statements cannot be made for more complex representations. As for light sources, current measurements typically do not yield spectral radiometric data. Additionally, influences like bending of underlying geometry, different temperatures and many more are not considered. For a truly predictive rendering simulation, this needs to be improved in the future.

## 12.2 Verification of Rendering Step

Compared to the verification of elements in a modeled scene, a large amount of research has been done to validate the correctness of light transport simulation methods. Existing approaches either focus on the creation of comparison data by specifying scenes with known analytic or measured light distributions, or on comparison of simulated data and ground truth through image metrics or psychophysical studies. In the following, results from these categories are described. Further information can be found in the state of the art reports by Ulbricht et al. [528] and McNamara [351], where the latter concentrates on image comparison metrics and psychophysical studies.

### Known Analytic Solutions

A convenient way for testing the accuracy of light transport simulation software is benchmarking against scenes with known light distribution. The most accurate way to specify such benchmarks is definition of scenes with analytically computable solutions.

The first such scenes were introduced by Hyben et al. [229]. They proved that the reflected radiance is constant in planar scenes with geometry representing the interior of a circle and constant, uniform reflectivity. By additionally emitting light from parts of the geometry a very simple benchmark scene can be created. The approach was later extended to 3D spheres [511].

More complex scenes with analytically computable solutions were presented by Smits and Jensen [490]. The five test scenarios allow testing of simple shadowing, and sup-

port diffuse and perfectly specular BRDFs. To remove the limitation to simple BRDFs and light sources, Szirmay-Kalos et al. [509] define requirements for BRDFs and light sources such that scenes of constant radiance are obtained. For a given geometric configuration of a closed scene the BRDF of surfaces can be adjusted to predefined light sources or the other way round. This allows efficient testing of importance sampling of light sources and BRDFs in Monte-Carlo raytracers. Another six test scenes suitable for verification of indoor illumination were proposed by Maamari and Fontoynont [331]. In comparison to previous benchmarks, the scenes are geometrically very simple and contain simple BRDFs only but they allow for simulation of transmittance and light sources with measured properties.

## Measured Samples

Since derivation of analytical solutions for complex scenes is extremely difficult or even impossible, many researchers instead favor scenes with measured solutions. Already in 1985, Cohen et al. [72] and Meyer et al. [362] measured the light distribution in the famous Cornell box by tabulating radiant energy flux densities at 25 points in the cube. To remove limitations to very simple scenes and simple lighting, Takagi et al. [512] measured luminance values at several points in a complex outdoor scene with an Incident Color Meter. The points were chosen such that numerous types of light-matter interaction were covered. Strangely, they did not use all measured points in a following comparison between reality and the results of a raytracing method. A more precise approach was taken by Mardaljevic [341]. He performed simultaneous measurements of skylight conditions and interior illuminance in two rooms with different glazing systems, known geometry and reflectivity. More accurate and complex measurements were performed by Drago and Myszkowski [115] who measured the illuminance at a large number of points in the atrium of the University of Aizu. High accuracy was achieved by providing a very exact scene description including measured BRDFs of the most common materials, calibrated light sources (including measured maintenance factors), and complex, accurate, modeled geometry. Concerning the combination of complexity and exactness, their setup can still be considered the best today. An even more accurate yet less complex scene was acquired by Schregle and Wienold [459]. In their setting, multiple sensors track the light source's luminance intensity and movable sensors inside a cube allow for fast illuminance measurements. The authors additionally provide a careful description of the calibration steps and perform a well-founded error analysis.

Another way of improving measurements was presented by Karner and Prantl [254]. In addition to measuring luminance values at few locations, they took a photograph of the scene and fit a curve mapping measured pixel colors to luminance values, which

results in approximate luminance values for a much larger number of measurement points. Unfortunately, camera parameters could not be reconstructed perfectly which leads to geometric misalignments between the camera image and images rendered from the scene. To compensate for this problem, luminance measurements were taken at points robust to alignment problems, and problematic areas in the photograph were omitted during comparison. A similar yet more elaborate approach based on CCD cameras was presented by Pattanaik et al. [407]. The authors presented a calibration scheme for CCD cameras such that accurate colorimetric data (i.e., CIE XYZ values) can be derived from several images taken with narrow-band filters.

In summary, definition of standard benchmarks for light transport simulation methods seems to be a highly desirable if not absolutely necessary task for the future. Although benchmark testing will never be able to prove the correctness of software since excessive testing of all possible cases is impossible, it gives a strong indication towards or against correctness. Definition of benchmark scenes with measured light distribution appears to be a more flexible approach than relying on scenes with analytically computable solutions. Despite the many advances made in acquiring such scenes, existing data does not suffice for validating predictive rendering since typical measurements do not consider different light spectra and since they contain irradiance values only. An additional problem is related to accurate replication of scenes. Probably a careful combination of the approaches of Drago and Myszkowski [115] and Schregle and Wienold [459] could result in a much more concise and complex data set than currently available.

## Image Comparison Metrics

Having scenes with known light distributions at hand, the correctness of light transport simulations can easily be validated by comparing simulated and reference values. If the values match, the algorithm passes the test. Otherwise, some parts of the simulation are flawed. Unfortunately, it seems very unlikely that any algorithm will pass such a test in the near future since too many influences are currently not modeled at all. Therefore, people want to quantify which degree of correctness some algorithm has.

Concerning radiometric correctness, this measure can be computed from root mean square (RMS) differences or peak signal to noise ratios (PSNR) between the correct and the simulated values or images. Since current algorithms are far from being able to computing such perfect solutions in interactive times, most researchers instead focused on perceptual comparisons of images. Such comparisons are motivated by the fact that most rendered images are observed by humans. Therefore, image differences can be tolerated as long as human observers cannot perceive them. While this strive for photometric correctness is less ambitious than the goal of predictive rendering (i.e.,

reproduction of radiometrically correct images), it appears to be a reasonable measure given the current state of technology.

Perceptually motivated image comparison metrics try to take into account as many properties of the human visual system (HVS) as possible (for an overview of relevant properties see [140]). Most of them are influenced by image compression techniques which try to remove unrecognized image detail and measure the likelihood of perceiving such detail removements by *just notable differences* (JND). JNDs thus represent local assessments of similarity.

Within the research areas of image compression and computer vision image comparison metrics have been studied for a long time and many results were achieved [58]. Among these, especially two received significant attention: the Sarnoff Visual Discrimination Model (VDM) [328], which operates entirely in the spatial domain, and Daly's Visual Difference Predictor (VDP) [85], which operates partially in frequency space. A study by Rushmeier et al. [443] revealed that the VDP performs significantly better than RMS measures since it models three significant properties of the HVS. First, the ability to sense relative luminance values rather than absolute ones for a large range of luminances. Second, the non-linear, yet not explicitly known brightness perception of luminance values. Third, the dependence of the HVS's sensitivity on the spatial frequency of luminance variations. These results were confirmed by experiments by Martens and Myszkowski [343] and Longhurst and Chalmers [326]. A study by Li et al. [309] showed that the VDP and the VDM perform similarly well but that the VDM is slightly more robust since it requires less recalibration.

Obviously, these two metrics are not perfect. Therefore, researchers proposed extensions and new metrics that considered other aspects of the HVS. Bolin and Meyer [40] extended the VDM to operate on colors instead of luminance values by taking color aberration into account. They additionally proposed a more efficient implementation such that integration into global illumination computation algorithms becomes possible. Another efficient model was proposed by Neumann et al. [378]: Instead of comparing values per-pixel they operate on randomly selected, rectangular regions with varying size. Additionally, they compute color differences with a metric useful for complex environments [377]. A more elaborate metric similar to the VDM was proposed by Ramasubramanian et al. [434]. It is based on separation of luminance dependent and spatially dependent processing where the spatial part can be precomputed, making it useful in combination with global illumination solvers.

Other examples handling colors are the approaches of Ferwerda et al. [142], which includes an additional measure for prediction of visual masking, and the model of Pattanaik et al. [406]. The latter is especially interesting since it merges color appearance models and models based on spatial vision. Compared to previous metrics, it includes

luminance and chromatic adaptation and can thus simulate visual acuity. Additionally, such measures are important for comparison of appearance on different output devices. Similar results were published by Fairchild and Johnson [134] but they additionally included temporal adaptation [135], making their approach suitable for quality assessment of animations or videos. Other examples of metrics suitable for animated images are the temporal extension of the Sarnoff JND [13] and DVQ [568].

Due to the unknown characteristics of the HVS and the widely varying notion of perceptual correctness, other researchers proposed very different metrics. Wang et al. [562, 563] conjecture that the HVS is highly trained to recognize structures and therefore suggest to explicitly include structural information. For a large number of images they show that their mean structured similarity index measure (MSSIM) yields superior results compared to the Sarnoff JND. This evaluation was confirmed by a comparison of the MSSIM and the VDP [54].

Sheikh et al. [473] take a statistical approach to the problem. They argue that natural images form a tiny subset of all 2D signals only, which can thus be described accurately by *natural scene statistics*. They model such natural data with Gaussian mixtures per channel, where channels may include spectral channels and orientations. The authors define image similarity based on the amount of distortion between two images which can be computed by comparing the contained information. According to their experiments, their metric achieves slightly better results than the MSSIM.

An even more abstract version of a perceptual metric is defined by Winkler [582]. His goal is to include sharpness and colorfulness criteria since these tend to influence a picture's appeal. While interesting for image generation tasks, such a metric is clearly not useful for evaluation of predictive renderings.

Another very abstract yet potentially much more useful metric was proposed by Ferwerda and Pellacini [143]. Instead of assessing whether a human notices a difference between a pair of images, the intention of their metric is assessment of whether the differences influence the observer's ability to achieve a task, i.e., to judge the suitability of a specific product during a design review. The authors remark that such a metric obviously depends on the task to be performed, which might turn out to be the key problem of this approach. In addition, these metrics might be even more user-dependent than perceptual metrics.

## Psychophysical Studies

While image comparison metrics have the advantage of being computable in a fast, automatic, and objective way, they only model certain parts of the HVS, leading them to imperfect judgments compared to human observers. In addition, current metrics

are very susceptible to image alignment problems. To overcome these problems, several psychophysical studies comparing rendered and real scenes have been conducted. Unfortunately, studies including human subjects require images to be shown to users. Quality assessments therefore always include characteristics of display systems.

Initial studies [168, 483] checked the accuracy of radiosity solutions computed for the Cornell box by asking people to compare the real scene and rendered images. Drago and Myszkowski [115] made a similar experiment with a much more complex scene: Subjects were asked to compare the degree of realism of photographs and rendered images after inspecting the real scene through a peephole for a short time. Hardly a surprise, photos were considered more real than rendered images, which has to be attributed partly to the necessary tone mapping step. To reduce preferences for obviously real scenes, Meyer et al. [362] placed cameras in front of the real setup and a TV screen showing the rendered image, and showed the camera images to the subjects. As a result, people could hardly distinguish the real and the virtual scene, but disturbing influences due to the reduced dynamic range in the rendered image and the characteristics of the TV screen and the cameras certainly influenced this judgment. An even more sophisticated device enabling unbiased comparisons of images from different sources is the *periscope* proposed by Longhurst et al. [327]. The device includes a mirroring system that allows switching between three different source images (i.e., the real scene, rendered images, and photographs) without disturbing influences. It was used in a study aiming at answering the question why photographs are perceived as more realistic than rendered images. The authors concluded that imperfectness due to scratches, dust and dirt leads to more realistic images than perfectly clean, rendered ones. These results are confirmed by results on comparisons between the rendering quality achievable with simple BRDF models and BTFs, which are presented in the following chapter. Rademacher et al. [430] additionally pointed out that the presence of soft shadows and rough surface textures increases the likeliness of an image being perceived as a photograph.

Another experiment trying to determine factors that lead to perceived realism was conducted by Stokes et al. [499]. They rendered images of indoor scenes containing various aspects or mixtures of global illumination only (i.e., direct lighting, diffuse, glossy, and specular indirect lighting). Subjects were asked to rank resulting images w.r.t. their degree of realism. Interestingly, full global illumination solutions were often ranked equally well as partial solutions. The authors therefore proposed a metric intended to predict which time-consuming parts of a global illumination solver can safely be omitted. Related metrics were already used to efficiently produce global illumination radiosity [423] or ray-tracing [58] solutions and were even applied to interactive rendering [116]. Obviously such metrics are not directly useful for predictive rendering. Nevertheless, similar metrics based on radiometric data might prove useful.

A much more fundamental study was performed by McNarama et al. [352] which tried to assess differences in lightness perception in real and virtual environments. Subjects were asked to match the lightness of objects to predefined categories in both real and virtual environments. The results showed similar results for both environments leading to the conclusion that rendered environments are capable of conveying lightness information accurately.

### Discussion

Despite the many efforts dedicated to verifying the correctness of light transport simulations existing methodologies do not suffice. Clearly, definition of suitable test cases plays an important role for the verification step, which is mirrored by a recent CIE technical report [133]. Yet, as mentioned above, benchmarking can only provide indications to whether a lighting simulations is correct. As shown in existing work, such correctness measures should ultimately be based on RMS or PSNR comparisons. For the current state of technology and due to the potential savings in computation time, perceptual metrics were proposed which try to mimic the human perception process. While existing approaches are successful to a certain extent, severe improvements can be expected due to novel research results on the functionality of the HVS from biology and psychology. Further improvements should be possible by employing more accurate methods for registering real and virtual images or by adding abstraction mechanisms to image metrics that ignore slight misalignments.

In summary, verification of predictive rendering methods is not possible today. Future research is required to devise extensive test cases and comparison metrics based on radiometric data to enable reliable statements concerning the degree of realism of predictive rendering software. Obviously, this also requires suitable methods for acquisition of scene properties. Fortunately, existing evaluations provide a strong indication that extensions of available methods might be well suited for radiometric rendering.

## 12.3 Verification of Display Step

Having verified the correctness of scene properties and the rendering method, the final step in evaluating the predictive image generation process is validation of the display step, e.g., on caves, powerwalls, monitors, or print media. Unlike previous steps, which can all be performed in radiometric units even today, current display technology is limited to a fixed number of base colors, i.e., photometric entities. While existing printers support up to seven different base colors, other display devices are typically limited to

three channels. To minimize loss of realism due to this limitation *gamut mapping* techniques were devised. A second fundamental problem of current display technology is limitation to a narrow band of displayable luminance (notable exceptions were recently proposed by Seetzen et al. [464, 463]). Reducing the high dynamic range (HDR) of lighting simulations to the greatly reduced dynamic range of display media is done by *tone mapping* algorithms, which typically take the non-linear luminance perception of the HVS into account. An overview over existing tone mapping techniques can, e.g., be found in the book by Reinhard et al. [436], the paper by Yoshida et al. [605], or in the recent paper of Krawczyk et al. [280]. Clearly, gamut and tone mapping have no influence on radiometric predictiveness. Yet, due to missing multispectral HDR displays their influence on current VR users needs to be considered.

A different approach towards validating the correctness of VR environments is based on Ferwerda's definition of functional correctness [141]: If the user is equally successful in performing a task in reality and VR, the VR environment should be considered functionally correct. Clearly, this approach differs significantly from predictive rendering but follows the same goal: Allowing the users to make optimal decisions. An example of a study evaluating this functional goal was performed by Ferwerda et al. [144]. The authors tried to assess the ability of VR users to perceive complex shapes by asking subjects to identify shapes in images rendered from different view-points and with either local or global illumination. Hardly surprisingly it turned out that global illumination simulations lead to better results since they resemble lighting in real environments much closer. A large number of similar experiments testing several other properties like the ability to navigate or concentrate in VR environments are described in the survey by O'Sullivan et al. [400]. They also describe other aspects influencing perceived realism like the effects of different animation techniques, the type of representation of the VR user, and the use of geometric level of detail.

## 12.4 Discussion

As described above, validation of predictive rendering techniques is a complex and difficult problem since it requires validation of all parts of the predictive image generation process, including interactions of different stages. While verification techniques exist for all parts of the process, most of them are not sufficient for qualitative assessment of predictive realism. Necessary improvements in the future include measurement of multi-spectral radiometric data, robust comparison metrics for radiometric data, and significant improvements of display devices since they impose the largest restrictions on the overall achievable quality at the moment.

# Chapter 13

# Evaluation of BTF Rendering Quality

In the previous parts of this thesis methods for interactive, physically-based high-quality rendering of scenes were described. Special attention was devoted to efficient handling of accurate, spatially varying materials, i.e., BTFs, since images generated with these methods clearly show increased realism compared to standard approaches. While the improvement of image quality seems to be obvious, the exact degree of realism achieved with BTF materials has not been quantified so far.

The previous chapter clearly pointed out that quantification of such improvements is necessary to achieve acceptance among possible users. Therefore, this chapter tries to answer especially two questions:

- To what degree does the use of BTFs instead of textures and simple BRDFs increase the degree of realism of a rendered image?
- How far are images with BTF materials from reality?

The answers to these questions are not only of interest for assessing the quality of current rendering methods but additionally have impact on future developments. Possible problems with BTF materials will guide future research into techniques for acquisition, synthesis, compression, and rendering of such materials. Problematic interactions with the rendering software might have an impact on the development of physically-based rendering software.

In the following, first the approach for answering these two questions will be described, then the necessary steps for validation of predictive images will be presented (i.e., scene and light distribution acquisition, rendering, and validation), and finally con-

clusions will be drawn from the evaluation. The contents of the chapter were published in [360] already.

## 13.1 Approach

Quality improvements due to BTF materials are difficult to measure especially if specialized interactive rendering algorithms are employed. Therefore special focus was layed on two aspects. Verification of the simulated global light distribution ensures that BTFs behave similar to BRDF material representations at a coarse scale. Verification of the local appearance of BTF textured surfaces additionally investigates the replication of material effects at a fine scale and thus evaluates if materials are recognized.

A second decision to be made for the evaluation is the choice of benchmark scenes. To facilitate industrial use of BTF materials, the scenes should represent industrially relevant scenarios, which implies complex geometries and complex lighting. To fulfill these requirements, the interior design process of a car was chosen as a representative example. The scene thus consists of a real car and several lighting environments that simulate different weather conditions and locations. The exact scene acquisition procedure is described in the following.

As a third decision one has to choose an appropriate rendering method. To minimize unforeseen influences, path tracing is used since it is known to produce physically accurate results and since it allows integration of almost arbitrary geometries, materials, and lighting environments. Necessary settings for the employed renderer (MentalRay in the presented case) are outlined below.

As a fourth decision it was decided that the assessment of quality improvements should be based on a comparison of photographs and rendered images despite the known problems of such an approach [430, 327]. This method was chosen since other techniques have problems with efficient replication of complex lighting environments. Details on the acquisition of photographs, the comparison procedure, and evaluation results are given below.

## 13.2 Scene Acquisition

In order to verify a rendered image all components contributing to the final image need to be acquired (i.e., light, material, and geometry) and the light distribution in the physical scene needs to be captured.

For the chosen scenario light source acquisition requires capturing HDR lighting environments. To accomplish this task, the technique of Debevec and Malik [95] was

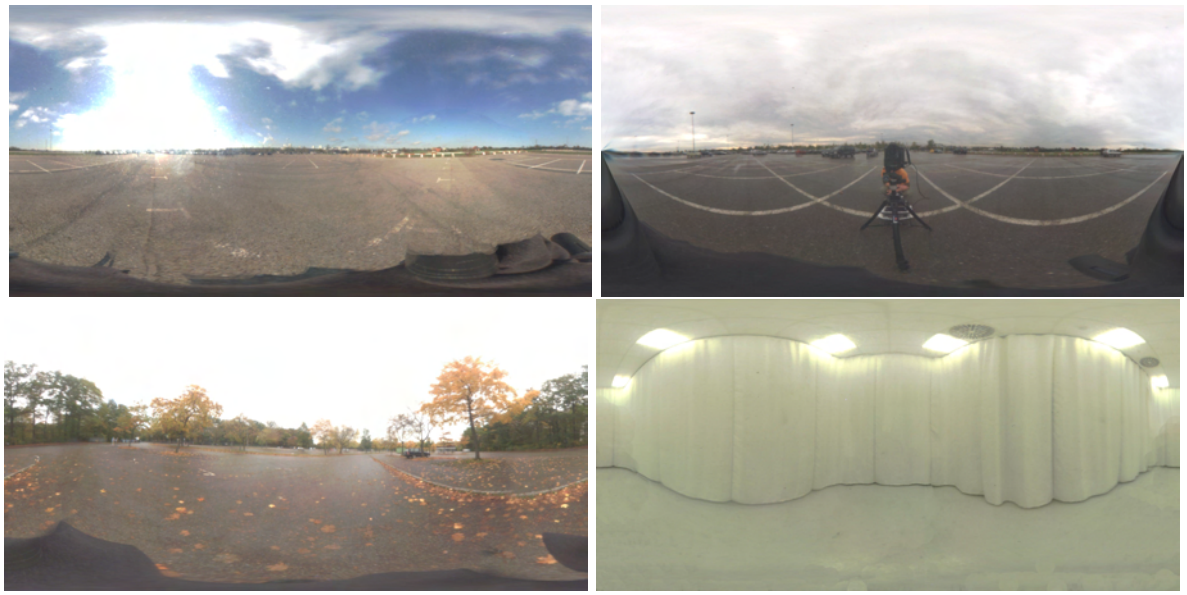


**Fig. 13.1: Acquired lighting environments. From top to bottom, left to right: sunny day and cloudy day on parking space, cloudy day on different parking space, and indoor environment with curtains.**

selected which generates HDR lighting environments from a series of pictures of a reflecting sphere taken with varying exposure times. For acquisition a Kodak DCS 760 Professional digital CCD camera was chosen which can be controlled remotely (thus minimizing jiggle artifacts) and allows exposure times between $1/8000$ s and several minutes. To account for the brightness and wavelength dependent sensitivities of the CCD chip, the camera's response curve was measured before each acquisition event by taking pictures of a white, diffuse surface.

Different lighting environments were acquired to simulate light distributions for different weather conditions and at various locations. Some of these are shown in Figure 13.1. They vary from sunny days, which results in environment maps with very high dynamic range and rendered images with strong specular effects and sharp shadows, over cloudy skies, which generate mainly diffuse impressions in resulting images, to indoor environments. The shown indoor environment is lit by six indirectly illuminating fluorescent tubes and thus features a very low dynamic range and mainly diffuse lighting. The environments were captured with the smallest shutter value available (f22) to avoid depth blur. Only for indoor scenes a shutter value of f11 was used to avoid much longer exposure times leading to significant image flaws due to temperature rise in the CCD chip.

During acquisition two main problems arose. First, images of the sunny sky featured oversaturated pixels due to the sun's intensity even at very short exposure times. Instead of removing these by a neutral density filter as suggested by Stumpfel et al. [501], the

sun is modeled as an area light source with constant radiant intensity which is derived from the physical properties of the sun and the modeled scene. To compensate for the various sources of light attenuation like scattering and reflection effects in the atmosphere, the intensity was reduced manually. Remaining errors are almost invisible to the observer due to the approximately logarithmic mapping of luminance values in the HVS, which is replicated in existing tone mapping operators. Additionally this modeling step turned out to be very useful in combination with path tracing since it leads to significantly improved importance sampling of the lighting environment.

The second problem is the movement of clouds. Their position is slightly shifted in photographs of varying exposure, leading to imperfect environment maps. Even worse, acquiring the light distribution in the car could not be done at the same time as acquiring the environment maps, leading to significant cloud movements. Therefore, great care was taken to capture light situations where cloud movement has hardly any effect on the light distribution in the car.

A second step for acquisition of scene properties is measurement of the reflectance properties of materials in the car (a Mercedes Benz C class model). As in the experiment of Drago and Myszkowski [115] only the most significant materials were acquired. Unlike for previous validation experiments, HDR material BTFs were acquired using the setup of the Universität Bonn, which is described in more detail in Chapter 3. To enable simultaneous handling of several BTFs, measured data was compressed using local PCA [369]. Figure 13.2 gives an overview of the acquired materials. For a later comparison between the rendering quality achieved with BTFs and standard materials, materials were also modeled by Phong BRDFs and textures. While the Phong BRDF parameters were provided with the car's geometry, the frontal views of the BTF materials (i.e., polar angle $0°$) were used as textures. To match the brightness of light reflected from the BTFs and the Phong BRDFs, Phong parameters were scaled by hand, iterating between parameter adjustment and rendering until the appearance matched approximately.

The third part to be acquired in the scene is the geometry. For the given test, the CAD NURBS data of a Mercedes C class model (stored as CATIA V5 model) was finely tessellated and stored as OpenInventor model. Materials were assigned to the surfaces using the Maya modeling software, and texture coordinates were computed using two different approaches. For surfaces covered with structured materials like cloth or wood, texture coordinates were computed using the method of Degener et al. [100]. For all other surfaces, texture coordinates were computed with the synthesis on surfaces method by Magda and Kriegman [334], which was implemented as a Maya plugin and which handles both standard textures and BTFs. It iteratively textures triangles such that textures follow a defined orientation field and such that minimal discontinuities are

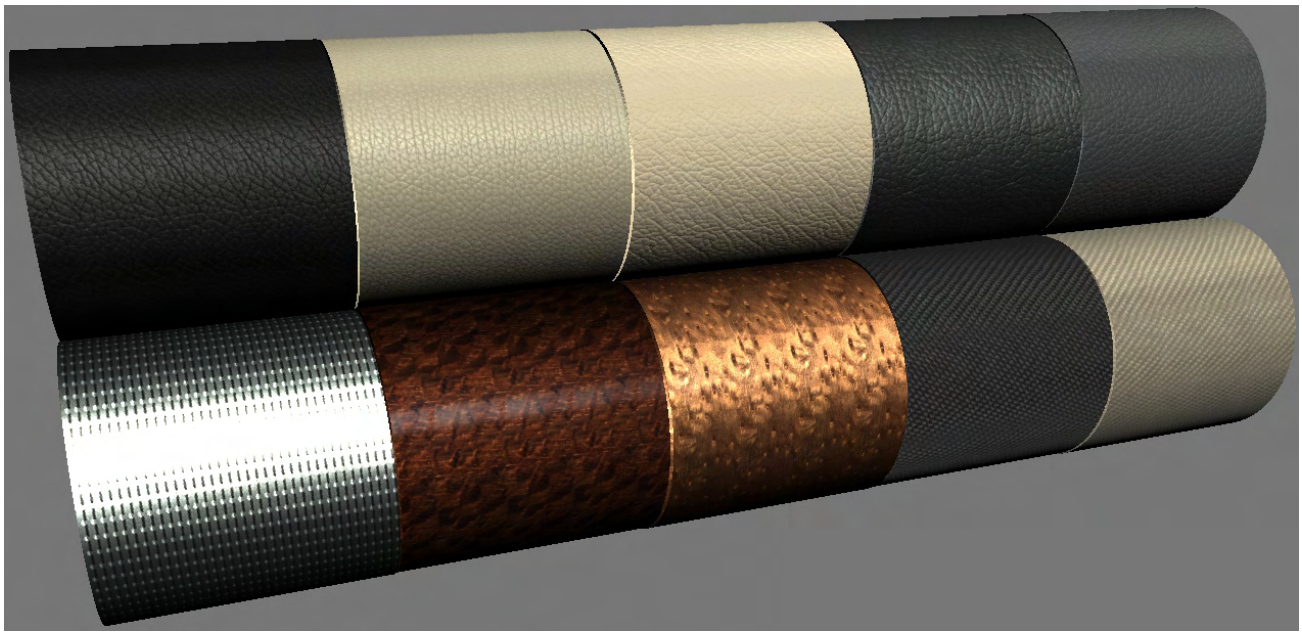


**Fig. 13.2: Acquired BTFs. From top to bottom, left to right: dark and light synthetic leather, light and dark natural, and dark synthetic leather, hard shell containing aluminium, two types of wood, and dark and light seat covers.**

visible at triangle edges. Necessary orientation fields can easily be specified using Maya interfaces. Due to bad connectivity information in the geometry (e.g., some neighboring triangles were not connected) which leads to difficulties when specifying complex orientation fields, it turned out that a single direction vector per geometric component is sufficient. Additionally, special care was taken that the triangles in the geometry were smaller than the texture patch. This restriction can be relaxed by applying standard texture synthesis algorithms to enlarge the base texture. Results of synthesizing textures on car interior surfaces are shown in Figure 13.3.

The fourth and final part of scene acquisition is measurement of the light distribution in the benchmark scenes. To enable comparison of large-scale and small-scale effects, pictures showing the complete cockpit and zoom-ins onto selected regions were taken. To register the photographs with the 3D geometry a standard registration procedure was employed which requires the user to select a number of corresponding points in the image and the 3D scene. Assuming a simple pinhole camera [203], at least six pairs of corresponding points need to be selected. Usually, registration gets more stable if more features are selected. Yet, as Figure 13.4 shows determination of exactly corresponding points is difficult due to lighting influences like shadows. Therefore, it turned out that selection of fewer (typically eight to ten) but well chosen pairs leads to better results than selection of more, possibly unreliable pairs. Since radial distortion is not modeled

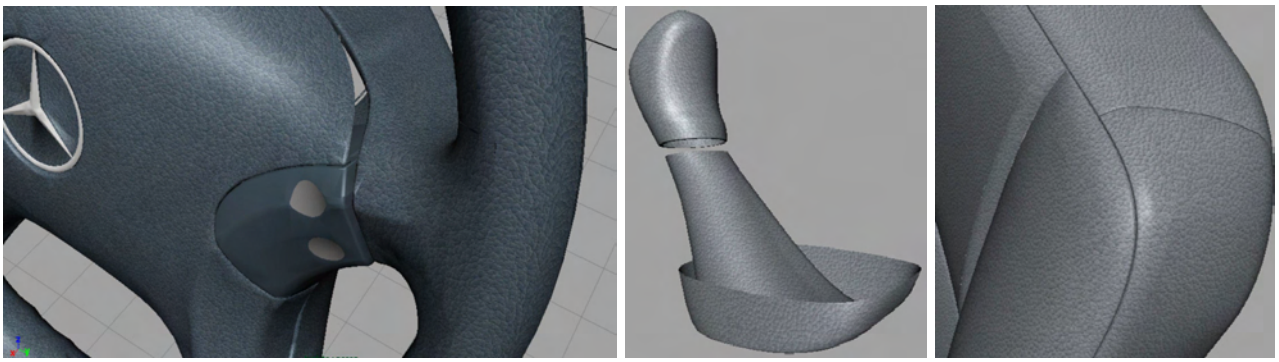


**Fig. 13.3: Objects textured by synthesis on surface algorithm.**

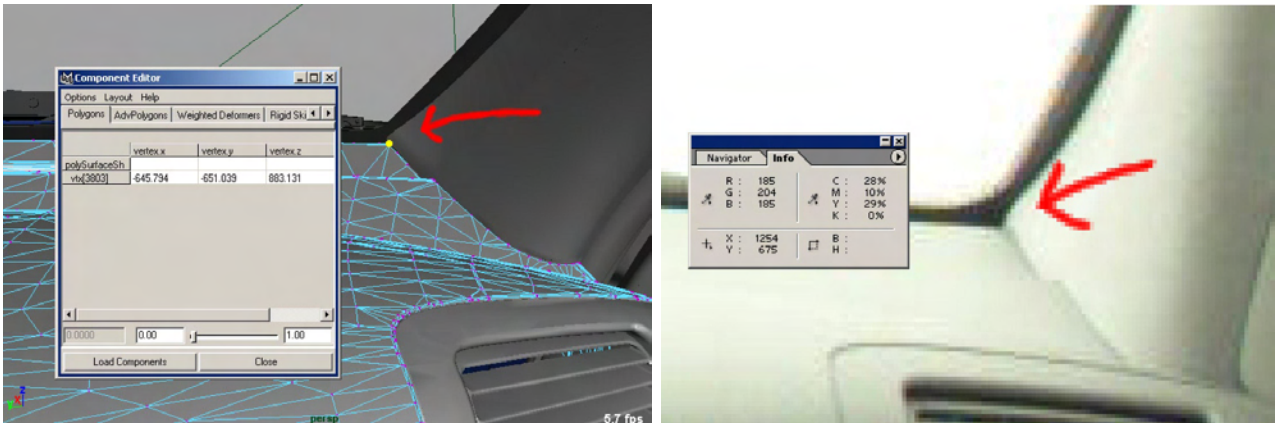



**Fig. 13.4: Registration of photograph (right) to 3D scene (left). The exact location of the highlighted vertex is difficult to determine in the photograph.**

in the simple pinhole camera model, it additionally turned out that feature points with varying depth values lead to significantly improved registration.

While registration of photographs showing the complete cockpit worked well, zoom-ins often did not contain a sufficient number of feature points. Therefore the following approach was taken: After capturing a zoom-in photograph, another picture was taken with all camera parameters but the zoom unchanged. This allows to determine camera parameters from the first image, which shows a significantly larger part of the cockpit, and only the zoom has to be determined afterwards, which can be done manually.

## 13.3 Rendering

In order to render the acquired scenes from captured view-points with MentalRay, BTF materials were integrated as a shader. Unfortunately, integrating these shaders with photon mapping turned out to be difficult. Therefore, path tracing was employed although

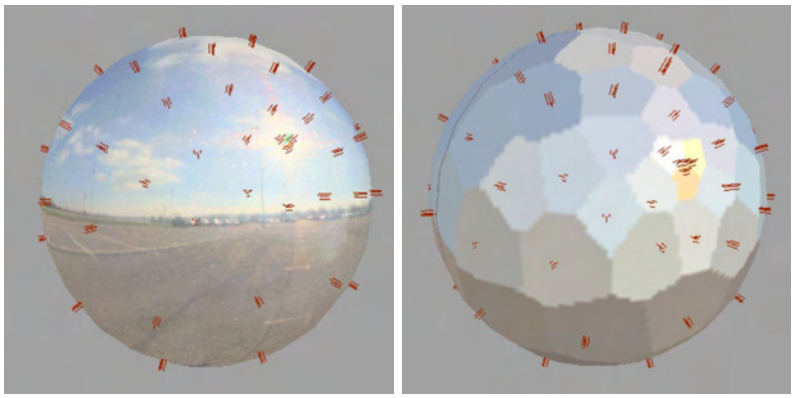


**Fig. 13.5: Light clustering. Left: environment map with cluster directions, right: cluster regions.**

photon mapping would be more efficient for environmental lighting. To compensate for this, lighting environments were approximated by a number of clustered directional light sources as proposed by Cohen and Debevec [69] and Kollig and Keller [276] which leads to largely reduced rendering time for noise-free images. Figure 13.5 shows an example of applying this technique to a sunny environment map. Since the chosen number of clustered light directions directly impacts rendering quality and rendering speed, determining an optimal number minimizes rendering time without impacting visualization quality. Figure 13.6 shows the rendered images using different numbers of light clusters for cloudy and sunny environments, and a relatively glossy material. For the cloudy environment, hardly any differences are visible between the images rendered with 16 and 32 clusters and 10 shadow rays per light source (the large number of shadow rays is used since MentalRay allows shooting of shadow rays into regions of directions which are determined from the clusters' Voronoi regions). Therefore, 64 clusters are chosen for cloudy environments. For the sunny sky, 128 clusters are necessary since differences start being imperceptible between 32 and 64 clusters.

Another practical problem to be solved w.r.t. the lighting environment is alignment with the geometry. Due to the sun's movement between shots taken for capturing the environment and the light distribution in the car, incorrect shading is visible for the sunny sky if the lighting environment and the geometry are aligned perfectly (i.e., if the environment is aligned such that the positions of objects like trees match the ones in the photographs of the car interior). Due to the known time between the two shots, the sun's approximate movement can be computed, resulting in an approximate rotation for the environment map. This rotation can later be improved by a non-linear optimization step matching the location of highlights.

To replicate the real camera, the virtual shutter's dimension needs to be set to $1.049 \times 0.695$ which was determined by experimenting.

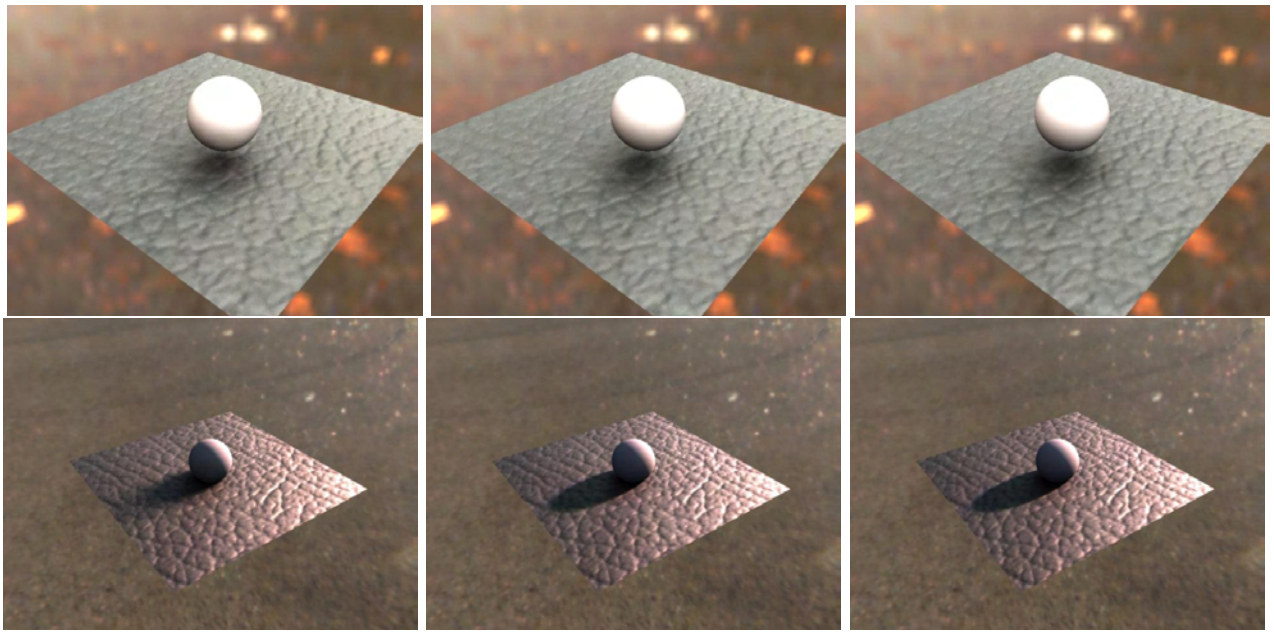


**Fig. 13.6: Influence of the number of light clusters on the resulting quality. Top: cloudy sky, bottom: sunny sky. Left: 16 light clusters, middle: 32, right: 64.**

A significant increase in rendering performance can be achieved for images showing parts of the car mainly influenced by direct lighting (e.g., the dashboard). In this case it is possible to delete all light sources from the lower hemisphere and to render direct illumination effects only. Additionally, invisible geometries can be removed if they do not occlude light sources. This reduces the total memory requirements and increases rendering speed.

Having rendered the images quite significant color differences between the rendered images and the photographs became visible. The reasons for these problems are analyzed below. To compensate them, a $3 \times 3$ color transformation matrix was determined by a non-linear scheme minimizing the least squares difference between selected regions of blurred images. Selected regions mask badly registered image parts and imperfectly replicated geometries.

## 13.4 Evaluation

As described in the previous chapter, evaluating the similarity of images should optimally be performed using suitable image similarity metrics, since they give an objective measure of quality. Unfortunately, this approach cannot be used for the given experiment since image metrics compute differences by comparing pairs of corresponding pixels. Establishing such pixelwise correspondences was not possible due to the fol-

lowing reasons:

- The geometries of the cars in the real and virtual scenes are not identical. While the CAD model represents a model from car design, the real car was slightly modified compared to the design version (e.g., central louvers in dashboard). Additionally, many textiles draped over the car's surface show pleats which are not modeled in the geometry (especially on the car seats and the doors).
- Replication of the exact small scale structure of materials in the real car is not possible since texture synthesis methods generate similar distributions of small scale features only.
- Lighting environments are neither perfectly aligned in photographs and rendered images nor of even resolution. Additionally, influences of the windshield are not simulated.
- Registration errors could not be eliminated completely.

### 13.4.1 Validation of Large Scale Effects

Fortunately, these problems can be circumvented by simple measures when evaluating the global light distribution in the car. To avoid influences of different geometries and lighting environments, these parts are removed from compared images. To compensate registration errors and different small-scale structure, the resolution of the images is reduced by a factor of 8. Finally images are transformed into grayscale to avoid color calibration artifacts, and their intensities are scaled to achieve equal average brightness.

Figure 13.7 shows resulting images for the car's dashboard. They were computed with direct lighting only. It is clearly visible that the structure of the lighting with BTFs resembles the original light distribution much closer than if using the Phong BTFs specified with the car model. This observation is underpinned by the HDR VDP [340] images comparing the original image and the rendered ones. The image rendered with BTFs shows significantly less regions where differences were determined (i.e., colored regions) and the differences are less perceivable on average. The VDP determined that in the image with BTF materials, about 3.41% of the pixels are perceived as being different at a probability greater 75%, and only 2.1% at a probability greater 95%. For comparison, the same thresholds apply to 5.78% respectively 3.87% of the pixels in the image with standard materials. These results show that BTFs lead to a more accurate light distribution than standard materials. Obviously, the parameters of the Phong model

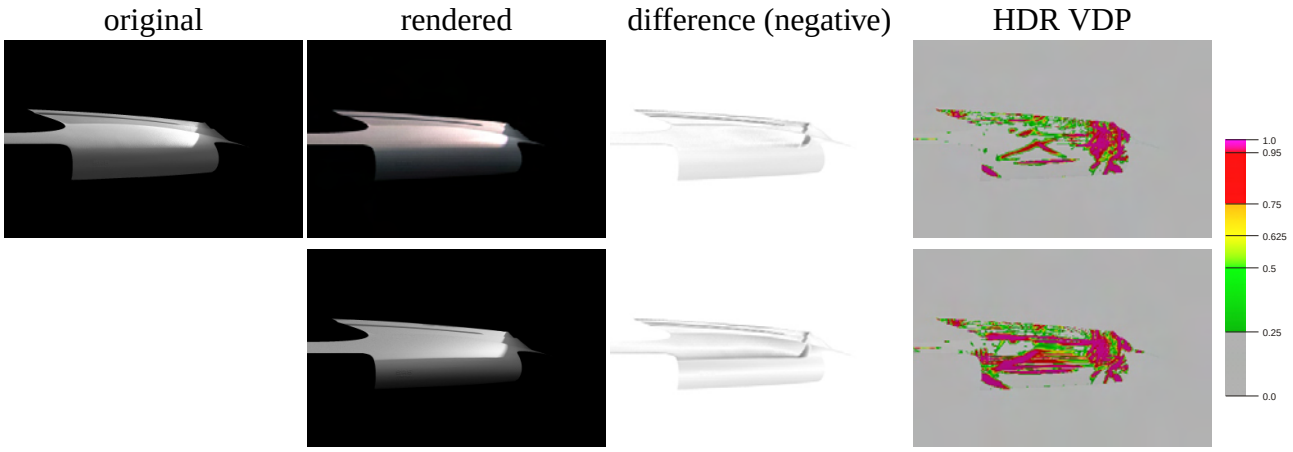



**Fig. 13.7: Global Light Distribution on dashboard. Top: BTF, bottom: standard materials.**

could be changed to optimize resulting highlights but it remains questionable if such a specific set of parameters would be suitable for other configurations of view and light.

Although measured BTFs achieve much more realistic highlights, the comparison image still shows significant errors. Most of these occur at edges and are due to registration errors. Another source of error is incorrect alignment of lighting as the shadow cast by the right A post shows. In order to quantify the errors solely due to the BTF a second experiment was conducted. Several images were rendered with the sunny sky and varying lighting parameters (i.e., the sun's intensity and the size of the region of directions probing the sun's visibility). Resulting images are shown in Figure 13.8. It is clearly visible that the shape of simulated highlights does not match the ones from corresponding photographs. Highlights in rendered images are less focused, yielding the impression of a less specular material. The deviation can be explained by two facts. First, materials in the real car are worn. Multiple touches might have turned them greasy which would explain the shinier behavior. Second, directional sampling of measured BTFs might have been too coarse, thereby diffusing highlights. Summarizing these observations, the errors due to BTF acquisition are very small compared to the errors introduced by other facts like light modeling, registration, and material changes. Therefore, the abilities of the BTF to simulate global light distribution can be considered very accurate.

### 13.4.2 Validation of Small Scale Effects

The previously described experiments showed that measured BTFs are superior to standard Phong BRDFs combined with diffuse textures for simulating global light distribution. Since this can be achieved by measured BRDFs as well, a second series of experiments aims at comparing the small scale appearance. Unlike for large scale eval-

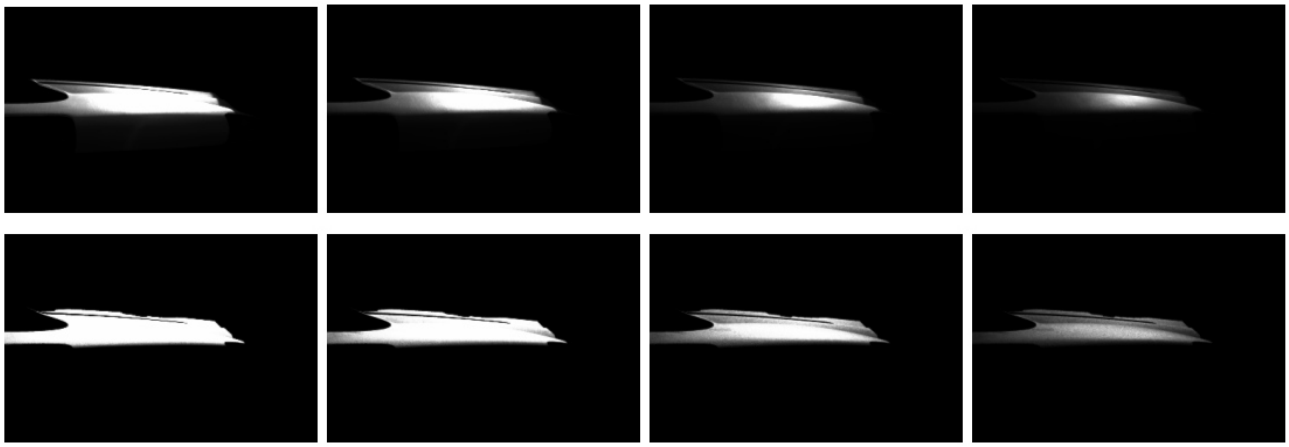


**Fig. 13.8: Structure of the highlight on the dashboard for varying exposure in the photograph (top). The bottom images show rendered images where the light's intensity was set such that the highlight's maximum intensity matches the one in the photograph.**

uations, registration errors and surface structure mismatches cannot be circumvented by reducing resolution since small scale effects need to be retained. Therefore, evaluation based on metrics is not possible in this case. Instead a study with human subjects was conducted.

As a preparation for this study, nine (regions in) photographs were selected (five with sunny sky, three with cloudy sky, two indoor) and rendered once with texture modulated Phong BRDFs and once with BTFs. To maximize the expressive power of the study, images focused on parts covered with measured materials. Additionally, to hide lighting environment resolution differences, visible parts of the outside in the rendered images were replaced by visible parts from the photographs. The 27 resulting images (cf. figures 13.13 and 13.14) were shown to the 22 subjects participating in the study in random order by a simple interface (cf. Figure 13.9) which additionally queries the subject's assessment of realism. Participants were told that some images are rendered but were otherwise naive to the goals of the study. All of them were no computer graphics experts and had normal or corrected to normal vision. To minimize the influence of presentation order and to quantify learning effects each participant was shown all images twice. The detailed results can be found in Tables 13.1-13.3.

A direct interpretation of the values is not suitable since all participants used a different range of numbers to judge the realism of images (cf. Figure 13.10). Therefore, the assessments are normalized according to the following formula

$$\tilde{X}_{i,j} = \frac{X_{i,j} - \mu_i}{\sigma_i}$$

with $X_{i,j}$ the assessment of the $j$th image by participant $P_i$, $\mu_i$ and $\sigma_i$ the average rat-

**Fig. 13.9: Interface for quality assessment.**

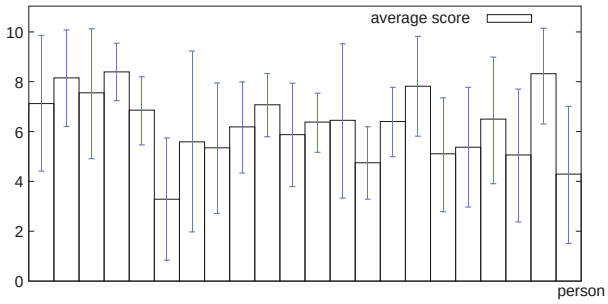



**Fig. 13.10: Average quality scores and their standard deviation per participant.**

ing and standard deviation by $P_i$. Normalization results in average scores of zero and standard deviations of one for all participants. The resulting normalized values are summarized in Figure 13.11. They show that, on average, BTF images scored almost as high as photographs. Both performed much better than images rendered with standard materials (Phong BRDFs and diffuse textures) which can also be seen in the average quality assessments of individual images. The performance of standard materials is especially bad in combination with the sunny sky, i.e., an environment with a dominant directional light, since directional lights tend to increase the visibility of surface structure due to sharp shading. Interestingly, photographs are not always considered most realistic which corresponds to the observations of Rademacher et al. [430]. In the given case this can possibly be attributed to two reasons. First, the photographs feature a slight blur which might lead to the impression of less structured materials, especially in combination with diffuse lighting environments (images 7, 8 and 9). Second, subjects were shown no reference image.

Another notable observation is that the perceived realism seems to depend on the shown section of the scene. If a photograph is given a high score, the quality assessments of corresponding BTF and standard images tend to be high as well. This is in correspondence with Rademacher et al.'s observations who showed that photographs of scenes with specific settings (e.g., soft shadows, rough surfaces) are judged more realistic than ones of other scenes. It also confirms that BTFs lead to consistently better results than standard materials.

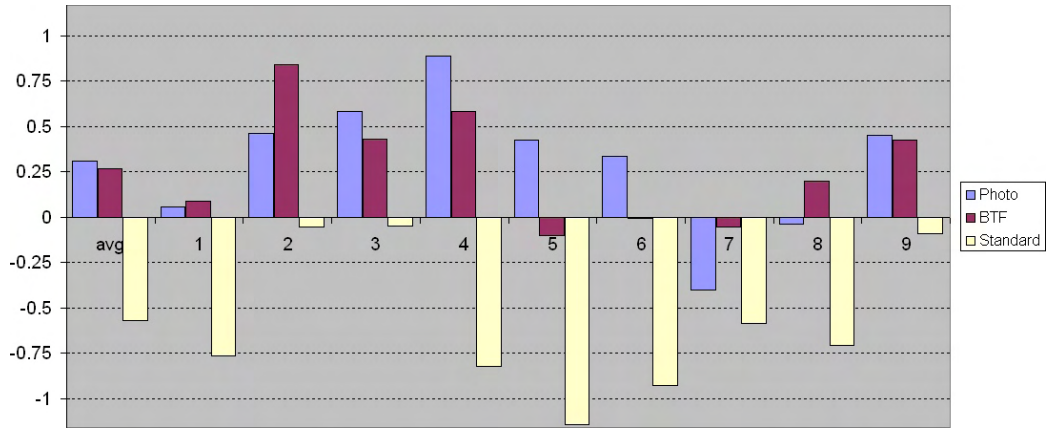



**Fig. 13.11: Average normalized scores for each image and generation method. Leftmost: Average normalized scores over all images.**

## 13.5 Conclusions

Evaluation of the rendering quality achievable with measured BTF materials has shown that BTFs offer a significant advantage over standard material representations. Like measured BRDFs, they are well suited for simulating global light distributions. Unlike measured BRDFs they additionally replicate small-scale surface features in a physically accurate way. At both scales they were found to be superior to standard material representations that combine diffuse textures and Phong BRDFs.

Unfortunately, the conducted studies were not able to measure the remaining error caused by BTF representations due to various reasons. Obviously, all steps contributing to predictive rendering possibly introduce errors. Known error sources include the following points:

- Lighting environments vary due to sun and cloud movement.
- Lighting environments are not perfectly aligned.
- The measured materials and those used in the real car vary. Materials in the real car are worn which is likely to influence their reflectance behavior. Determination of the exact influence of aging remains for future work.
- Not all materials in the car were measured. Quality assessments were thus always influenced by geometric parts covered with Phong BRDFs.
- Compression of measured BTFs introduces errors, the amount and type depending on the compression technique.

- The real and virtual geometry are not identical. This includes differences in the way materials are applied to surfaces. While perfect, fixed application is possible in VR, real materials often cause pleats and are thus movable to some extent.
- The texture synthesis method might be ill suited for anisotropic BTFs. Additionally, texture synthesis from relatively small texture patches cannot replicate the full complexity of materials like leather.
- Influences of the windscreen were omitted, which turn out quite significant for some parts of the car.
- Computed images could not be registered perfectly and contained significant color changes. Better results could be achieved in more controlled environments and with better calibrated cameras. Color shifts might additionally be caused by the material of the reflecting sphere used for capturing environmental lighting.
- Some errors, although probably small, were introduced by the imperfect light simulation software.
- Presenting the images to human subjects required tone mapping, which introduces significant errors as well. Additionally, images represent mappings of reality only. More concise results might be achieved by comparing rendered images and real models.

Another limitation of this study is the restriction of Phong BRDFs to the ones specified with the digital model. The experiment of Drago and Myszkowski [115] showed that specifically, artistically modeled Phong BRDFs may lead to more realistic images than measured BRDFs, which might also be the case for BTFs. Yet, considering the goal of predictive rendering it has to be mentioned that Phong BRDFs cannot be optimized for all combinations of viewpoint and lighting. Optimal parameters for one case will turn out bad for other settings, which is clearly not desirable for predictions. Therefore the restriction to given parameters seems to be a reasonable choice.

A third limitation is due to simulating direct lighting in the rendered images only. While this impacts the degree of realism in images lit by the sunny sky just slightly, influences on scenes with mainly diffuse lighting are difficult to judge, but probably significant. Figure 13.12 shows that quite different results can already be achieved by introducing approximate indirect lighting. Additional problems are introduced when simulating the lighting in buildings since the limited indoor distances are not well represented by environment maps, suppressing the significant spatial dependence of incoming light.

**Fig. 13.12: Changes in rendered images due to approximated indirect light. Left: direct lighting only, right: an additional area light source simulating indirect light in the interior.**

Despite the various sources of errors and the other limitations of this study, it was clearly shown that BTF materials substantially improve the realism of rendered images. Since this experiment focused on high-quality rendering with path tracing methods, which require offline rendering, this study can only provide an upper bound for the rendering quality achievable in VR. To evaluate the actual quality of rendering techniques useful for VR further experiments need to be conducted. In addition to evaluating the loss of quality due to real-time restrictions and the influences of animated scenes instead of static images, such experiments need to determine the influence of restricted resolutions of the display devices. Compared to BRDF material representations, the obvious advantage of BTFs is accurate reproduction of small-scale surface structure, which will only be visible in VR if high-resolution displays like the HEyeWall [281] are employed or if users zoom onto specific regions.

## 13.6 Future Work

The observations from this study give a clear indication that BTF materials are useful in predictive rendering applications since they increase the degree of realism significantly. Obviously the conducted experiment focused on few topics while leaving out several others. One of these is the comparison of rendering quality from BTFs with the one from bump-mapped BRDFs where the color is modulated by standard textures. The inclusion of bump-mapping effects makes small-scale structure clearly visible which might compensate many limitations of BRDF material representations, especially for materials like the marked leather covering the car's dashboard.

Another experiment could quantify the impact of BTF materials much closer by utilizing a more controlled test environment: placing planar material probes in an envi-

ronment with controlled lighting (e.g., rooms with artificial lighting) after measuring their reflectance properties. Such a scenario allows for direct computation of similarities by image metrics since registration errors could be avoided since perfectly corresponding surface structures would be available, and since disturbing influences could be minimized. Influenced by comparison metrics for texture synthesis algorithms [112], existing image comparison metrics could be optimized to tolerate registration errors or different surface structures. Additionally, human subjects could be asked to perform a visual comparison of the real scene and the rendered image. Finally, using techniques to reproduce complex, changing lighting environments (e.g., [94]) even simulation of outdoor scenes would be possible.

Several studies should concentrate on the use of BTF rendering in VR. As a first step, simple animations (e.g., using varying lighting environments) could be produced and users could be asked to assess the degree of realism in comparison with videos of the real scene. Of course, this requires consistent capturing of interior photographs and environmental lighting, and requires highly accurate registration. As a second step, animations could be computed in real-time, thereby evaluating the capabilities of real-time BTF rendering methods.

A further experiment could focus on the user's ability to recognize a material from the rendered image by making the user guess the tactile properties of the rendered material. A setup for such an experiment could be as follows. The BTFs of several materials with different tactile properties are acquired. During evaluation, subjects are shown rendered images of objects (e.g., tori or spheres) covered with BTFs. Optimally these images are rendered in real-time to allow interactive adjustment of user position. Participants are additionally given the opportunity to touch several measured materials without seeing them. If the users succeed in matching rendered images to touched samples then obviously BTF rendering is capable of communicating the surface structure very well, which seems to be of significant importance for application areas like the cloth industry or interior design.

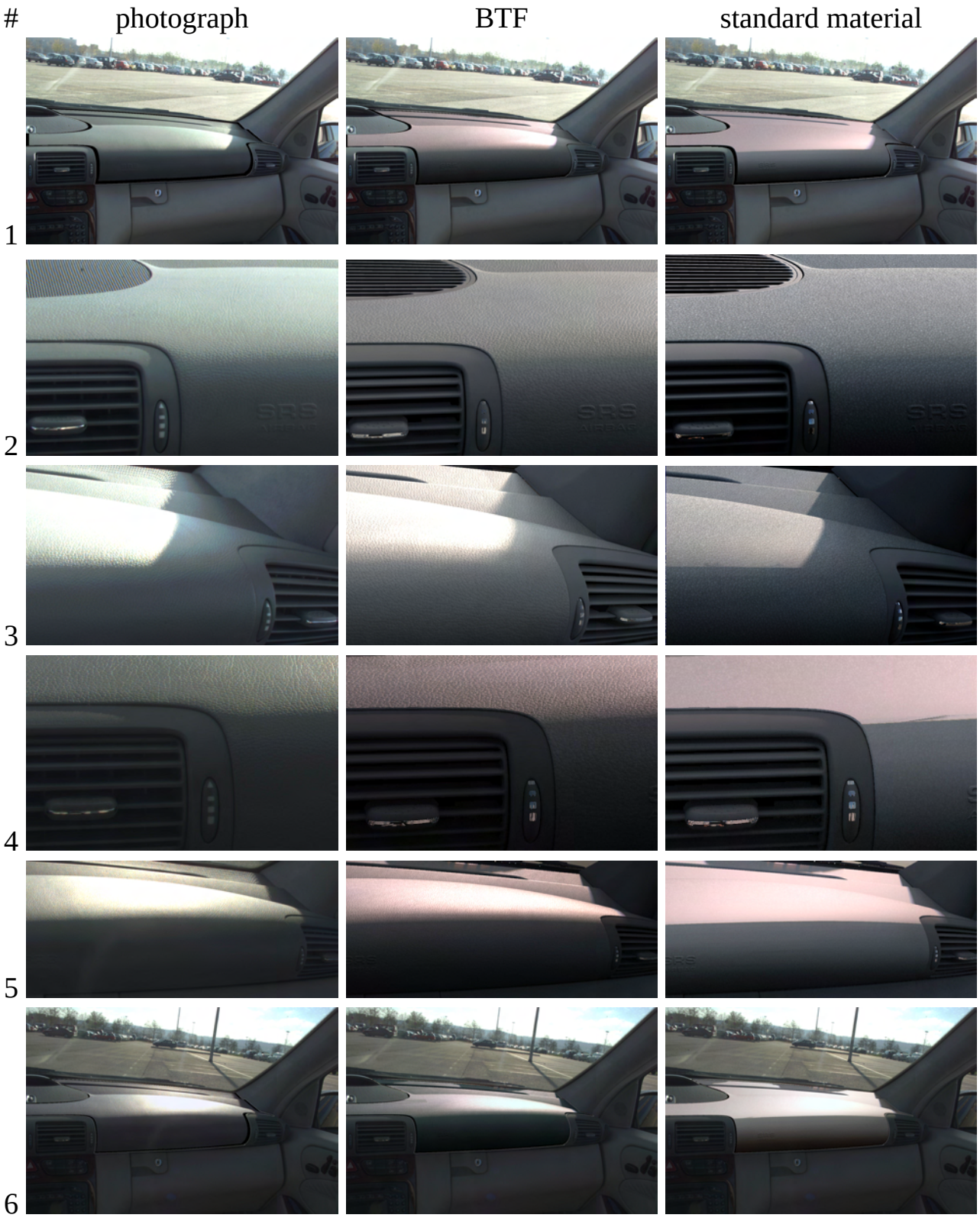



**Fig. 13.13: Images used in the psychophysical study (part 1).**

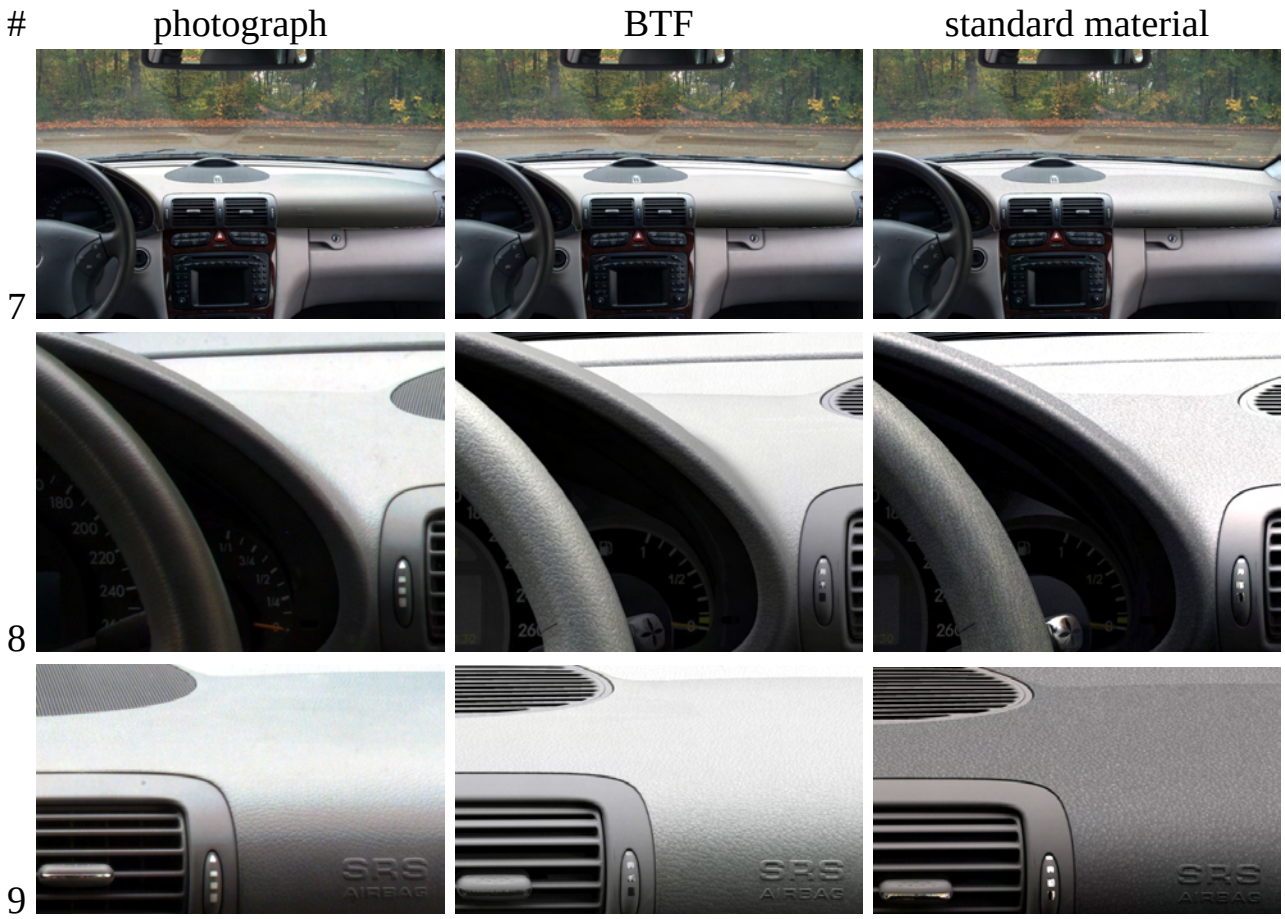



**Fig. 13.14: Images used in the psychophysical study (part 2).**

| image | person | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| P1 | 9 | 7 | 8 | 7 | 6 | 8 | 1 | 9 | 9 | 7 | 6 | 6 | 1 | 7 | 7 | 6 | 6 | 6 | 8 | 10 | 9 | 8 |
| | 9 | 9 | 3 | 7 | 6 | 1 | 1 | 6 | 6 | 7 | 6 | 5 | 1 | 6 | 7 | 5 | 5 | 5 | 9 | 8 | 9 | 9 |
| P2 | 10 | 10 | 10 | 10 | 9 | 6 | 10 | 2 | 4 | 9 | 8 | 8 | 9 | 6 | 5 | 10 | 6 | 4 | 8 | 10 | 10 | 5 |
| | 9 | 10 | 10 | 10 | 9 | 3 | 8 | 2 | 8 | 7 | 1 | 8 | 9 | 3 | 5 | 10 | 7 | 1 | 8 | 8 | 8 | 5 |
| P3 | 10 | 10 | 10 | 8 | 8 | 2 | 9 | 8 | 6 | 8 | 8 | 8 | 9 | 2 | 9 | 9 | 9 | 7 | 9 | 4 | 10 | 2 |
| | 10 | 10 | 10 | 9 | 9 | 5 | 8 | 8 | 7 | 9 | 6 | 8 | 8 | 3 | 7 | 9 | 8 | 4 | 8 | 5 | 9 | 5 |
| P4 | 10 | 10 | 9 | 10 | 8 | 4 | 8 | 9 | 4 | 9 | 8 | 8 | 9 | 6 | 7 | 10 | 9 | 6 | 9 | 10 | 10 | 6 |
| | 10 | 10 | 8 | 10 | 8 | 3 | 10 | 8 | 9 | 8 | 6 | 8 | 9 | 6 | 7 | 10 | 9 | 8 | 8 | 10 | 10 | 4 |
| P5 | 8 | 9 | 10 | 8 | 6 | 3 | 8 | 8 | 3 | 8 | 8 | 8 | 8 | 4 | 6 | 9 | 8 | 7 | 7 | 6 | 8 | 4 |
| | 8 | 10 | 7 | 10 | 8 | 5 | 9 | 7 | 4 | 9 | 6 | 7 | 8 | 5 | 7 | 7 | 9 | 6 | 7 | 10 | 9 | 6 |
| P6 | 9 | 8 | 8 | 7 | 6 | 1 | 1 | 9 | 5 | 6 | 8 | 7 | 8 | 6 | 8 | 9 | 5 | 10 | 9 | 8 | 10 | 1 |
| | 10 | 10 | 10 | 8 | 6 | 5 | 1 | 7 | 8 | 8 | 7 | 7 | 1 | 6 | 8 | 6 | 5 | 8 | 9 | 6 | 10 | 10 |
| P7 | 6 | 6 | 10 | 7 | 7 | 8 | 1 | 4 | 7 | 9 | 6 | 5 | 4 | 4 | 8 | 5 | 4 | 3 | 5 | 4 | 7 | 2 |
| | 4 | 6 | 9 | 8 | 7 | 9 | 1 | 3 | 4 | 8 | 5 | 4 | 1 | 4 | 8 | 5 | 2 | 3 | 9 | 3 | 6 | 1 |
| P8 | 7 | 5 | 10 | 8 | 6 | 3 | 10 | 5 | 8 | 8 | 7 | 6 | 9 | 7 | 4 | 10 | 2 | 9 | 5 | 6 | 8 | 5 |
| | 4 | 6 | 8 | 7 | 9 | 1 | 10 | 3 | 7 | 7 | 6 | 7 | 7 | 6 | 6 | 10 | 2 | 3 | 2 | 6 | 6 | 3 |
| P9 | 10 | 10 | 10 | 10 | 6 | 1 | 8 | 8 | 7 | 8 | 3 | 7 | 9 | 4 | 9 | 10 | 8 | 6 | 7 | 3 | 8 | 5 |
| | 8 | 10 | 9 | 9 | 9 | 2 | 10 | 6 | 8 | 8 | 4 | 6 | 9 | 4 | 9 | 10 | 7 | 3 | 6 | 8 | 7 | 9 |

**Tab. 13.1: Quality assessment for the photographs. Top row: first run, bottom row: second run.**

| image | person | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| B1 | 10 | 10 | 6 | 9 | 7 | 1 | 2 | 7 | 7 | 7 | 7 | 6 | 4 | 6 | 5 | 8 | 3 | 7 | 8 | 4 | 10 | 8 |
| | 9 | 10 | 7 | 7 | 7 | 7 | 1 | 7 | 5 | 6 | 10 | 7 | 1 | 7 | 6 | 4 | 5 | 8 | 4 | 2 | 10 | 8 |
| B2 | 10 | 10 | 10 | 10 | 9 | 8 | 8 | 9 | 8 | 8 | 10 | 7 | 9 | 5 | 7 | 10 | 6 | 6 | 9 | 3 | 10 | 2 |
| | 10 | 10 | 10 | 10 | 8 | 8 | 8 | 9 | 5 | 8 | 8 | 7 | 9 | 7 | 7 | 10 | 8 | 7 | 8 | 8 | 10 | 4 |
| B3 | 10 | 10 | 8 | 8 | 8 | 1 | 7 | 9 | 5 | 8 | 8 | 8 | 9 | 5 | 7 | 7 | 9 | 8 | 9 | 5 | 10 | 1 |
| | 8 | 10 | 10 | 9 | 8 | 5 | 4 | 5 | 5 | 8 | 6 | 8 | 9 | 5 | 8 | 6 | 8 | 2 | 9 | 9 | 8 | 3 |
| B4 | 10 | 10 | 9 | 8 | 6 | 1 | 8 | 6 | 9 | 8 | 8 | 7 | 9 | 5 | 7 | 10 | 6 | 6 | 7 | 7 | 10 | 4 |
| | 9 | 10 | 8 | 10 | 8 | 5 | 8 | 9 | 8 | 9 | 6 | 7 | 9 | 6 | 7 | 8 | 4 | 4 | 8 | 10 | 10 | 6 |
| B5 | 10 | 10 | 9 | 8 | 6 | 1 | 8 | 6 | 3 | 7 | 4 | 8 | 7 | 3 | 5 | 8 | 3 | 7 | 8 | 4 | 8 | 2 |
| | 6 | 9 | 7 | 9 | 7 | 1 | 1 | 7 | 7 | 7 | 7 | 7 | 9 | 5 | 6 | 8 | 4 | 2 | 6 | 3 | 8 | 4 |
| B6 | 9 | 7 | 8 | 7 | 7 | 1 | 1 | 8 | 6 | 8 | 1 | 6 | 4 | 4 | 8 | 7 | 7 | 7 | 8 | 8 | 10 | 10 |
| | 8 | 7 | 10 | 7 | 5 | 3 | 1 | 8 | 4 | 7 | 7 | 6 | 1 | 3 | 8 | 6 | 7 | 8 | 9 | 3 | 10 | 7 |
| B7 | 9 | 8 | 8 | 8 | 8 | 4 | 1 | 7 | 6 | 6 | 4 | 5 | 3 | 7 | 8 | 5 | 6 | 7 | 8 | 5 | 9 | 4 |
| | 8 | 8 | 10 | 8 | 7 | 1 | 1 | 3 | 8 | 7 | 5 | 4 | 1 | 6 | 7 | 5 | 6 | 7 | 8 | 7 | 10 | 3 |
| B8 | 9 | 10 | 9 | 9 | 7 | 3 | 9 | 6 | 6 | 8 | 5 | 6 | 6 | 5 | 7 | 8 | 5 | 8 | 4 | 3 | 7 | 2 |
| | 7 | 10 | 8 | 9 | 7 | 1 | 9 | 4 | 8 | 6 | 6 | 6 | 9 | 7 | 7 | 10 | 7 | 10 | 2 | 5 | 7 | 7 |
| B9 | 7 | 9 | 10 | 9 | 6 | 5 | 8 | 9 | 4 | 7 | 4 | 7 | 8 | 5 | 8 | 9 | 7 | 8 | 9 | 5 | 9 | 4 |
| | 8 | 10 | 10 | 9 | 8 | 1 | 8 | 7 | 6 | 6 | 7 | 8 | 9 | 6 | 8 | 9 | 5 | 4 | 9 | 4 | 10 | 8 |

**Tab. 13.2: Quality assessment for the images with BTFs.**

| image | person | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| S1 | 4 | 5 | 8 | 7 | 7 | 2 | 1 | 4 | 4 | 6 | 5 | 6 | 5 | 6 | 5 | 6 | 3 | 7 | 8 | 4 | 7 | 1 |
| | 2 | 6 | 9 | 7 | 8 | 2 | 1 | 3 | 4 | 5 | 6 | 4 | 1 | 4 | 5 | 4 | 2 | 3 | 6 | 2 | 4 | 3 |
| S2 | 7 | 8 | 4 | 10 | 7 | 1 | 9 | 4 | 8 | 6 | 4 | 5 | 8 | 4 | 6 | 9 | 5 | 7 | 8 | 4 | 10 | 2 |
| | 4 | 7 | 3 | 10 | 6 | 1 | 8 | 3 | 8 | 7 | 7 | 7 | 9 | 6 | 5 | 10 | 4 | 4 | 4 | 2 | 10 | 7 |
| S3 | 3 | 7 | 3 | 9 | 6 | 5 | 9 | 3 | 8 | 6 | 8 | 6 | 9 | 5 | 6 | 10 | 5 | 2 | 8 | 2 | 9 | 2 |
| | 4 | 7 | 3 | 9 | 7 | 6 | 8 | 4 | 8 | 7 | 6 | 7 | 9 | 5 | 6 | 10 | 3 | 3 | 4 | 4 | 10 | 8 |
| S4 | 6 | 9 | 2 | 7 | 2 | 3 | 5 | 2 | 4 | 7 | 1 | 7 | 6 | 3 | 6 | 7 | 4 | 2 | 1 | 4 | 6 | 1 |
| | 2 | 5 | 4 | 7 | 5 | 3 | 8 | 2 | 8 | 5 | 7 | 5 | 8 | 2 | 5 | 7 | 4 | 4 | 1 | 3 | 8 | 5 |
| S5 | 3 | 5 | 5 | 7 | 5 | 1 | 1 | 2 | 4 | 5 | 4 | 6 | 9 | 2 | 5 | 7 | 3 | 6 | 4 | 1 | 2 | 1 |
| | 2 | 5 | 5 | 7 | 7 | 2 | 3 | 1 | 4 | 5 | 6 | 5 | 8 | 2 | 4 | 5 | 2 | 2 | 1 | 1 | 5 | 1 |
| S6 | 9 | 5 | 8 | 7 | 5 | 3 | 1 | 3 | 4 | 5 | 6 | 6 | 6 | 5 | 5 | 6 | 2 | 8 | 7 | 3 | 3 | 1 |
| | 2 | 5 | 7 | 7 | 6 | 2 | 1 | 2 | 4 | 4 | 5 | 5 | 1 | 3 | 5 | 5 | 3 | 5 | 2 | 3 | 5 | 1 |
| S7 | 7 | 6 | 8 | 8 | 4 | 9 | 6 | 2 | 4 | 7 | 5 | 4 | 5 | 4 | 4 | 6 | 6 | 8 | 2 | 4 | 7 | 4 |
| | 4 | 7 | 9 | 8 | 7 | 1 | 1 | 2 | 8 | 8 | 7 | 5 | 1 | 4 | 5 | 5 | 2 | 3 | 3 | 3 | 6 | 2 |
| S8 | 4 | 5 | 2 | 8 | 6 | 1 | 9 | 4 | 6 | 5 | 3 | 5 | 4 | 4 | 5 | 10 | 2 | 3 | 8 | 3 | 8 | 1 |
| | 3 | 6 | 3 | 10 | 6 | 1 | 10 | 2 | 7 | 5 | 1 | 6 | 6 | 4 | 5 | 8 | 2 | 1 | 4 | 2 | 8 | 4 |
| S9 | 7 | 9 | 4 | 9 | 7 | 5 | 6 | 5 | 9 | 8 | 8 | 6 | 9 | 2 | 5 | 10 | 4 | 4 | 5 | 6 | 10 | 1 |
| | 3 | 8 | 3 | 9 | 6 | 3 | 8 | 2 | 7 | 6 | 6 | 6 | 9 | 5 | 5 | 9 | 2 | 3 | 7 | 2 | 10 | 9 |

**Tab. 13.3: Quality assessment for the images with standard materials (textures and Phong BRDFs).**

# Part VI

# Conclusions and Future Work

# Chapter 14

# Conclusions

## 14.1 Summary of Thesis

The focus of this thesis is elaboration of the importance of predictive rendering for several Virtual Reality applications, reviewing and presenting existing and novel technology that contributes to achieving this goal, and identification of missing techniques. As a framework for the technologies, the predictive image generation process of the RealReflect project (cf. Figure 1.2) was chosen since it represents the first approach towards an integrated solution. The chapters of this thesis were devoted to reviewing and describing the techniques required to implement the stages of this process, including in-depth descriptions of own contributions. Due to the huge amount of work required for implementing the full process, the personal work contributed to selected stages only, bearing a focus on modeling and real-time rendering of scenes with BTF materials.

In the following, a very brief summary of the treatment of the stages of the predictive image generation process is given. Following the partitioning approach of the verification framework of Greenberg et al. [175], the process tasks are separated into modeling, rendering, and display stages.

### 14.1.1 Scene Modeling

Accurately modeling scenes lays the basis for generating predictive images. If the scenes do not contain radiometrically correct descriptions of entities, the generated images must fail to be truly predictive. This thesis followed the standard modeling approach, which divides scene objects into light sources and geometric objects, which themselves consist of large scale geometry and small- and medium-scale material.

Light source modeling was covered just very briefly in this work. State of the art approaches acquire near-field properties of real-world light sources, capturing a very

accurate representation of the light emission properties. Unfortunately, most rendering techniques currently employ far-field properties only.

Modeling approaches for materials were devoted a large portion of this thesis. First, reviews of existing techniques showed that the most accurate material representations can currently be derived from measuring reflectance properties of existing materials. Then, techniques for efficient handling of such data were described, including a novel algorithm for compression of BTFs based on a parametric model and new approaches for synthesizing diffuse textures and BTFs.

For explicitly modeled geometry, existing representations relevant to Virtual Prototyping were summarized and necessary postprocessing steps for CAD data were described. A special focus was placed on level of detail representations, since they enable efficient and antialiased rendering, including own contributions for LOD representations of NURBS models and objects textured with complex material properties.

### 14.1.2 Rendering

Having adequately modeled scenes at hand, rendering algorithms transform the scene description into an image description showing the scene as seen by a camera or person. For predictive image generation, these algorithms have to take into account very accurate optics models and global illumination effects.

Therefore, an extensive review of existing rendering techniques suitable for interactive rendering of global illumination solutions was presented. Among the various kinds of algorithms, a combination of two techniques was identified to yield best-possible results for most walkthrough and design review applications given todays hardware technology: Rendering algorithms visualizing precomputed global illumination data yield good results for materials with low-frequency reflection behavior, effects due to high-frequency materials (i.e., reflections, refractions, and caustics) are added by separate approaches. Mirroring this finding, own contributions to these approaches were presented.

Since current VR systems are typically limited to local illumination models, moving to global illumination requires significant changes to existing, successful systems. To allow for more easily integrable solutions, various variants of a rendering technique for compressed BTFs were proposed, supporting different kinds of light sources.

### 14.1.3 Display

Displaying the results of the rendering algorithms to a human user is currently the weakest point of predictive image generation. In this thesis, current standard display technol-

ogy was identified to feature too low display resolution and dynamic range, and largely imperfect color reproduction. Fortunately, mirroring current trends towards high dynamic range rendering, recent developments improved the dynamic range of display devices. Other developments based on tiled, multi-projector displays largely enhanced display resolutions. Yet, so far no satisfying solution is in sight.

To compensate for these shortcoming, rendered images are fit to the characteristics of displays and the human visual system using tone mapping algorithms. A brief review of existing techniques showed that quite elaborate and successful approaches exist today. Yet, they cannot fully compensate the weaknesses of display hardware.

### 14.1.4 Verification

Due to a large number of limitations in the modeling, rendering, and display steps, the displayed images fail to predict reality. To assess the usefulness of generated results and to measure the remaining errors, all stages of the image generation process and their interplay need to be verified. As pointed out, this is especially necessary to overcome the existing and possibly well-justified negative attitude of designers towards Virtual Reality simulations.

A large number of existing verification attempts was surveyed, ranging from approaches verifying the individual stages to assessments of the quality of the entire image generation process. In addition, a study assessing the improvement of rendering quality due to measured BTFs compared to standard materials was described. It certified that more complex material properties lead to better image quality, which justifies the additional efforts required to support such complex materials, including the time invested into this thesis.

## 14.2 Implementing a Predictive Image Generation System

Due to the inherent complexity of the technology underlying each of the stages of the image generation process, the chapters of this thesis described developed techniques in a relatively isolated way. This approach seems suitable since the complexity of a system integrating all these components becomes immense. To nevertheless give an indication of the gain achieved by implementing a predictive image generation process, in the following the exemplary implementation of the RealReflect project is described briefly.


**Fig. 14.1: Light and material selection in the RealReflect system.** **(image courtesy of IC:IDO GmbH)**

The various modeling steps concerned with acquiring or modeling and processing light sources, materials, and geometry were implemented as stand-alone solutions or plugins for existing modeling packages. The tools included light acquisition and simplification, BTF acquisition, compression, and synthesis, and geometry conversion from CAD data, including parameterization and BTF synthesis on surfaces. Resulting data was made available to the VR application using a file format based on extending X3D scene descriptions [570].

Scene components are arranged using a scene editor implemented in a prototype VR system of IC:IDO GmbH. Figure 14.1 shows a screenshot of the system demonstrating interactive selection and placement of measured light sources and BTFs for a scene containing a Mercedes C class car model. The scene editor is integrated with a real-time renderer supporting complex light sources and compressed BTFs to give an instantaneous feedback of the look of the scene. It can additionally trigger offline precomputations of global illumination solutions, which can afterwards be inspected. Figure 14.2 shows an example where global illumination was precomputed for the front seats using a PRT approach (i.e., the lighting environment can interactively be rotated or exchanged). Note the strong difference in appearance compared to the back seats, which are lit by a point light source centered at the viewer's position. Rendered images are tone-mapped before displaying them. Great care was taken to realize tone mapping in an efficient way using GPU implementations.

Evaluations of the resulting system showed that the new components lead to significantly improved quality of rendered images although they are obviously neither photorealistic nor predictive. Additionally, due to the huge computational power of current of-the-shelf graphics boards, the rendering speed is well interactive to real-time (15-25 fps at least), resulting in good usability. In summary, the system proved to be a valuable

**Fig. 14.2: Views of a car where the front seats are covered by measured BTFs and lit by the environment.**

extension of existing VR systems.

Unfortunately, this extension does not come without a cost. Obviously, much more rendering power has to be spent for rendering BTFs than for standard materials consisting of Phong BRDFs, color and bump textures. Fortunately, this is hardly a problem given today's GPUs. The much more significant problem are increased efforts for modeling suitable scenes. Experience showed that several existing modeling approaches are not accurate enough if applied in combination with complex lighting and complex geometry. Among these are:

- Methods for computing texture coordinates
  In combination with diffuse textures, imperfectly chosen texture coordinates often remain invisible. This is often not the case in combination with BTFs. Figure 14.3 shows an example for texture coordinates generated by a synthesis on surfaces algorithm [334].

- BTF synthesis methods
  As shown in Chapter 4 BTF synthesis methods produce view- and light-dependent artifacts that are not present for diffuse color textures. In order to judge the suitability of a synthesized BTF, many combinations of view and light directions have to be checked.

- Local coordinate system generation
  Unlike Phong BRDFs, which are isotropic, BTFs typically show anisotropic be-

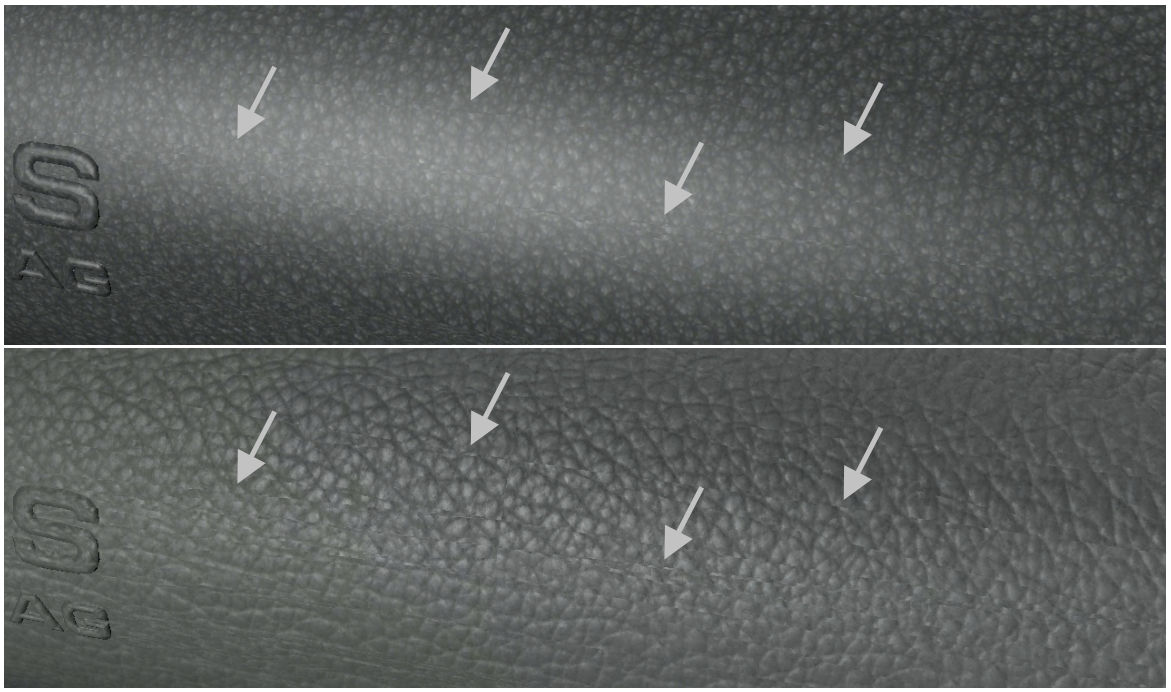


**Fig. 14.3: Imperfectly chosen texture coordinates lead to much more visible artifacts if the surface is textured with BTFs (bottom) instead of with diffuse textures (top). Images are zooms of those in Figure 2.21.**

havior. Therefore, inconsistent or poorly chosen local coordinate systems lead to false material appearance (cf. Figure 14.4).

- Light and material acquisition
  As Chapter 13 pointed out, the accuracy of the rendered image strongly depends on the sampling rate of the BTFs and the light sources. Choosing perfect sampling densities is a difficult problem.

Possibly, many other problems exist. The reason why these errors become more visible when employing correct light sources and materials might be that the rendered images appear to be more realistic than with simple lights and materials. Thus, observers will more easily notice sources of errors due to the larger visual difference between realistically looking regions and areas featuring disturbance of this realism. As a consequence, users of such potentially improved rendering software need to take care that the users do not get the impression that the rendering quality decreased due to more visible errors.


**Fig. 14.4: Change of appearance due to inverted bitangent in local coordinate system. Phong BRDFs typically used in VR applications lack such anisotropic behavior.**

## 14.3 Remaining Limitations

Existing implementations of predictive image generation processes that aim at rendering results at real-time frame rates need to make compromises between preprocessing time and memory, rendering speed, and achieved accuracy. Targeting real-time rendering of complex scenes, large compromises w.r.t. rendering accuracy have to be made. The RealReflect implementation, e.g., sacrificed the following effects:

- Volumetric Effects
  Materials were modeled with unique thickness, neglecting the volume of objects. In addition, volumes outside explicitly modeled objects were assumed to be evacuated and thus free of any participating media.

- Full Global Illumination Support
  Rendering from precomputed PRT data assumed distant lighting. Additionally, the combination of rendering algorithms (SLF or PRT rendering for almost diffuse, and interactive reflections for specular materials) does not cover the full spectrum of materials. Finally, global illumination effects were limited to static scenes.

- Accuracy limited to Modeling Accuracy
  As an example, rendering does not consider multi-spectral effects, which are es-

sential for many real-world situations.

In addition to the inaccuracies introduced to achieve increased rendering performance, other sources cause further errors. Among those are:

- limited accuracy of acquired lights and materials due to
  - acquisition errors,
  - undersampling,
  - RGB color model instead of multi-spectral measurements,
  - limited dynamic range,
  - time-invariance, and
  - planar materials (omission of bending/stretching effects),
- introduction of errors during data processing due to
  - reconstruction errors from compressed materials,
  - imperfect texture coordinate generation algorithms,
  - imperfect texture/BTF synthesis algorithms,
  - reconstruction errors from simplified light sources, and
  - inaccurate LOD representations for geometry (and material).

Although the rendering quality achieved for carefully modeled scenes is much higher than with standard approaches, designers currently refuse to use the VR solutions due to the above restrictions. The main reason seems to be the imperfectly reproduced colors of materials which is more easily noticed by trained users, who additionally pay special attention to these differences. Nevertheless, designers already started to accept offline versions of the image generation process that follow the same principle. As a result, such approaches are currently being introduced in the design process of several major companies. Extrapolating this development, it seems likely that due to novel computer hardware and improvement in rendering algorithms interactive implementations will become accepted in the next years.

# Chapter 15

# Future Work

## 15.1 Possible Future Trends in Graphics Hardware

The future development of computer graphics is deeply coupled with future advancements in graphics hardware. While initially growth in CPU processing powers was the driving factor, introduction of 3D graphics boards with an increasing number of units supporting dedicated graphics tasks changed orientation towards GPUs. The focus on graphics processors became even more significant with the introduction of freely programmable vertex and fragment processing units. Their introduction lead to a large number of novel rendering algorithms directly integrated with GPU rendering pipelines of existing graphics APIs, ranging from support for high-level surface primitives [480, 184] over complex light and material shaders [534] to full-featured raytracers [428].

Today rendering techniques show a wide spectrum of CPU and GPU use, ranging from CPU raytracing over rendering algorithms balancing CPU and GPU use to pure GPU-based rendering.

For the future, different trends for the development of graphics hardware are to be expected. Within the last few years the growth of computational power of CPUs has almost ceased to depend on the increase of clock frequencies due to physical limitations [506]. Instead, dual-core CPUs start becoming the standard and it seems very likely that this trend will lead to multi-core CPUs, practically yielding single-chip clusters of CPUs. This development should definitely push parallelizable algorithms like raytracing. In parallel, alternatives to existing CPUs are proposed: Quantum [438] or molecular [2] computers promise massively parallel processing power but estimation of their impact on computer graphics is currently not possible.

In contrast to CPUs, modern GPUs already feature a large number of parallel but simple processing units. Improvements in the last years have raised their computational

power over the level of comparable CPUs[1]. It seems very likely that future improvements will continue this trend by increasing the number of parallel units. In addition, it can be expected that future GPUs will be more flexible, featuring less restricted access to memory, increased instruction sets, and more programmable units[2]. This development will enable a large number of novel rendering techniques. In addition, it will improve opportunities for using GPUs as general-purpose stream processors [202] with computational power potentially far higher than the one of comparable CPUs.

Besides traditional CPUs and GPUs, new chips uniting ideas from both processing units have appeared and will be introduced in the future. As an example, the Cell processor [34] consists of a central processor resembling standard CPUs and a number of distributed co-processors resembling programmable GPUs, which are allocated for compute-intensive tasks. Such general-purpose processors are especially interesting for applications like games that simultaneously need to execute graphics, sound, physics, artificial intelligence, and other compute-intense tasks. An example of task-specific hardware units are raytracing cards [587]. They resemble existing GPUs but additionally feature unlimited recursive calls and memory accesses as existing CPUs. Due to their task-specific design they yield optimal performance, but also significant difficulties due to introducing hardware opposing the current main stream of rasterization-based rendering are to be expected.

Another important factor for graphics applications are memory access times, especially if models are handled that are too large to fit into main memory. Within the last 30 years the computational power of CPUs has increased at a much higher rate than memory access times for internal memories like RAMs and especially external memory like hard disks [336]. To bridge this gap, large on-chip caches were introduced, which improve the problem significantly as long as no cache misses occur. In order to avoid the large timing overheads for RAM or hard disk accesses caused by cache misses, the memory access patterns of future rendering algorithms need to be optimized carefully. Especially problematic w.r.t. this problem are Monte-Carlo global illumination solvers based on ray-tracing, since higher-order rays lead to almost random memory access patterns. A solution to this problem might be turning computational problems into stream processor problems which can be solved efficiently on, e.g., GPUs. Streaming programs arrange data in such a way that it is already ordered correctly, allowing efficient streaming of data through a pipeline of processing units, leading to local, predictive memory accesses.

[1] [202] states that in 01/2004 the peak performance of GPUs was almost 9 times as high as the one of comparable CPUs. In addition, a growth rate of factor 60 per decade for CPUs faced a growth rate of 1000 for GPUs.

[2] The upcoming Direct3D® 10 system [38] will support so-called geometry shaders, which enable programmable primitive setup. Due to the market power of Direct3D this feature will most likely be supported by GPUs soon.

## 15.2 Future Work on Computer Graphics Software

Independently but definitely influenced by the kind of compute hardware available in the near future, various developments in computer graphics software need to be made to permit (interactive) predictive image generation. In the following, these are grouped into tools for efficient scene modeling, more efficient rendering techniques, and rendering algorithms leading to higher accuracy due to inclusion of features not (commonly) supported so far.

### 15.2.1 Tools and Systems

In the previous chapter, it was shown that predictive image generation requires much more careful scene modeling than using traditional techniques. To make this process practical, novel modeling tools and techniques need to be devised and implemented.

As experience from the RealReflect project shows, many isolated techniques for efficiently modeling specific aspects of scenes suitable for predictive image generation exist already. Nevertheless, most of them are neither robust nor general enough for use in industrial scenarios. Therefore, they have to be improved and need to be integrated into an enclosing software system. Following these suggestions, very powerful tools could be developed, fulfilling the necessities of current predictive image generation processes while simultaneously allowing efficient use by trained users.

A special complication of such an implementation would be due to the large number of experts with very different expertise that would need to cooperate on such a task. Necessary fields of knowledge include system design, software engineering, acquisition and display technology, modeling, and rendering. Gathering such a number of experts might be possible in an industrially motivated project between industry and research institutes.

### 15.2.2 Efficient Image Generation

In addition to improving the modeling process, rendering techniques have to be enhanced. As mentioned in the previous chapter, a large number of rendering effects had to be sacrificed to achieve real-time performance. Obviously, this problem should be resolved in the future.

Depending on the type of rendering method employed in the future, which might either be rasterization-based as in RealReflect or raytracing-based as in [538], the following improvement need to be achieved:

- LOD techniques
  To reduce the memory footprint during rendering, minimizing the number of memory accesses, and reducing aliasing problems, LOD techniques are indispensable for future rendering systems. Future LOD representations need to take surface or volumetric properties like material into account in a more efficient way than existing approaches.

- Occlusion culling techniques
  For rasterization-based techniques, suitable occlusion culling methods need to be employed to handle the huge number of processed graphics primitives.

- Data management strategies
  To handle models too large to fit into main memory and to achieve coherent memory access, future rendering techniques need to place a specific focus on out-of-core data management. This concept should not be limited to data management on hard disk but all levels of the elaborate memory management hierarchy (e.g., prefetching schemes for data stored in RAM)

- Coherence exploitation
  Exploitation of spatial or temporal coherence is known to significantly increase performance in all application areas of computer graphics. Consequent use of this principle in rendering algorithms should therefore be mandatory in the future.

- Full support for global illumination
  Existing interactive schemes fail to handle scenes with all-frequency materials and all-frequency lighting. Bridging the gap between interactive rendering techniques with limited application area and limited accuracy, and very general, highly accurate but slow Monte-Carlo raytracing techniques will remain one of the major challenges for the future. Especially important in this respect are studies that try to predict necessary sampling rates for regions of specific scenes [117].

- Dynamic scene support
  Most rendering techniques presented in this thesis are limited to static scenes. Since VR needs to include the possibility to interact with objects, support for dynamically changing scenes is highly desirable.

- Parallelization
  Due to the current trends in hardware development, rendering algorithms should profit from even more parallelization. While raytracing techniques are relatively easily parallelizable, rasterization algorithms are mostly controlled by a monolithic process.

### 15.2.3 Accurate Image Generation

Implementing the proposed improvements will eventually lead to very accurate, real-time rendering algorithms. Nevertheless, to achieve truly predictive rendering, modeling and rendering have to be extended in a much more drastic way.

For the near future, the development of novel modeling or acquisition techniques for more accurate light sources, surface materials, and volumetric materials can be expected. Nevertheless, such technology additionally needs to feature shorter measurement times and minimized costs to achieve widespread use of this kind of technology.

Furthermore, more accurate color and optics models need to be used than today. First results for multi-spectral rendering have been published already [102], other approaches include wave-optics effects like interference [211, 375, 497] or polarization [579, 47]. Nevertheless, these concepts are not used in commercial products today, although at least multi-spectral rendering leads to quite different results for a large number of practically relevant scenarios. Obviously, to reproduce these effects, accurate light source and material representations need to be developed.

Finally, significant improvements in presentation quality need to be achieved by enhanced display technology. Future displays need to reproduce colors in a more accurate way, probably using more than three base colors like existing printers. Additionally, higher dynamic ranges are required and the dependence between the surrounding lighting and the colors displayed on the device should be examined and taken into account.

While improvement of rendering quality and speed will be the definite goal for future research, other efforts need to concentrate on verification of potential improvement. Following the approach of Greenberg et al. [175] all stages of the predictive image generation process should be verified individually, followed by a validation of the complete process. Especially helpful in this context would be the standardization of acquisition, modeling, and rendering techniques and processes, and required formats for exchanging data. Such a procedure would allow the definition of a number of standardized test cases and thus rigorous and efficient automatic validation.

Given the goal of predictive rendering, the validation should of course be made w.r.t. radiometric units. For the moment, this goal seems very ambitious. Therefore, fulfilling photorealistic correctness should be the current target. As a consequence, acquisition, modeling, and rendering should pay more attention to perceptual properties of the human visual system (HVS). To quantify the often unknown relevance of visual properties to the HVS, cooperations with psychologists and biologists should be undertaken. As shown in Chapter 12, such insights will not only increase image quality but can additionally reduce the amount of work for creating images by predicting onto which parts the efforts should be spent.

# Bibliography


[1] Salim S. Abi-Ezzi and Srikanth Subramaniam. Fast Dynamic Tessellation of Trimmed NURBS Surfaces. *Computer Graphics Forum*, 13(3):107–126, 1994.

[2] Leonard Adleman. Computing with DNA. *Scientific American*, 279(2):54–61, 1998.

[3] Daniel G. Aliaga, Jonathan Cohen, Andrew Wilson, Eric Baker, Hansong Zhang, Carl Erikson, Kenny Hoff, Tom Hudson, Wolfgang Stuerzlinger, Rui Bastos, Mary Whitton, Fred Brooks, and Dinesh Manocha. Mmr: An interactive massive model rendering system using geometric and image-based acceleration. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 199–206, 1999.

[4] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. Recent Advances in Remeshing of Surfaces. Technical report, AIM@SHAPE Network of Excellence, 2005.

[5] John Amanatides and Andrew Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *Proceedings of Eurographics*, pages 3–10, 1987.

[6] Carlos Andújar, Pere Brunet, Antoni Chica, Isabel Navazo, Jarek Rossignac, and Alvar Vinacua. Computing Maximal Tiles and Application to Impostor-Based Simplification. *Computer Graphics Forum*, 23(3):401–410, 2004.

[7] Thomas Annen, Jan Kautz, Frédo Durand, and Hans-Peter Seidel. Spherical Harmonic Gradients for Mid-Range Illumination. In *Rendering Techniques*, pages 331–336, 2004.

[8] Ian Ashdown. Thinking Photometrically Part II. In *LIGHTFAIR 2001 Pre-Conference Workshop*, March 2001.

[9] Ian Ashdown, David L. DiLaura, John Mardaljevic, Holly E. Rushmeier, Robert A. Shakespeare, Kenneth E. Torrance, and Greg J. Ward. Near-Field Photometry: Measuring and Modeling Complex 3-D Light Sources. In *ACM Siggraph 1995 Course Notes - Realistic Input for Realistic Images*, pages 1–15. ACM, 1995.

[10] Michael Ashikhmin. Synthesizing Natural Textures. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[11] Michael Ashikhmin and Peter Shirley. An Anisotropic Phong BRDF Model. *Journal of Graphics Tools*, 5(2):25–32, 2000.

[12] Michael Ashikmin, Simon Premože, and Peter Shirley. A Microfacet-Based BRDF Generator. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 65–74, 2000.

[13] ATIS. Objective Perceptual Video Quality Measurement Using a JND-Based Full Reference Technique. Technical Report T1.TR.PP.75-2001, Alliance for Telecommunications Industry Solutions, 2001.

[14] David H. Bailey and Paul N. Swarztrauber. The Fractional Fourier Transform and Applications. Technical Report RNR-90-004, NASA Advanced Supercomputing, Mail Stop T27-A, Moffett Field,CA 94035, April 1990.

[15] Kavita Bala, Julie Dorsey, and Seth Teller. Radiance Interpolants for Accelerated Bounded-Error Ray Tracing. *ACM Transactions on Graphics*, 18(3):213–256, 1999.

[16] Kavita Bala, Bruce Walter, and Donald P. Greenberg. Combining Edges and Points for Interactive High-Quality Rendering. *ACM Transactions on Graphics*, 22(3):631–640, 2003.

[17] Ákos Balázs, Michael Guthe, and Reinhard Klein. Efficient trimmed NURBS tessellation. *Journal of WSCG*, 12(1):27–33, 2004.

[18] Ákos Balázs, Michael Guthe, and Reinhard Klein. Fat Borders: Gap Filling For Efficient View-dependent LOD NURBS Rendering. *Computers and Graphics*, 28(1):79–86, 2004.

[19] David C. Banks. Illumination in Diverse Codimensions. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 327–334, 1994.

[20] Ziv Bar-Joseph, Ran El-Yaniv, Dani Lischinski, and Michael Werman. Texture Mixing and Texture Movie Synthesis Using Statistical Learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, 2001.

[21] Gill Barequet and Subodh Kumar. Repairing CAD Models. In *IEEE Visualization '97*, pages 363–370, November 1997.

[22] R. Basri and D. W. Jacobs. Lambertian Reflectance and Linear Subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):218–233, 2003.

[23] Rui Bastos and Wolfgang Stürzlinger. Forward Mapping Planar Mirror Reflections. Computer Science Technical Report TR98-026, University of North Carolina at Chapel Hill, 1998.

[24] Barry G. Becker and Nelson L. Max. Smooth Transitions between Bump Rendering Algorithms. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 183–190, 1993.

[25] Stephan Behrendt, Carsten Colditz, Oliver Franzke, Johannes Kopf, and Oliver Deussen. Realistic Real-Time Rendering of Landscapes Using Billboard Clouds. *Computer Graphics Forum*, 24(3):507–516, 2005.

[26] Jesse Bennett and Alireza Khotanzad. Maximum Likelihood Estimation Methods for Multispectral Random Field Image Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 537–543, 21 1999.

[27] Carsten Benthin, Ingo Wald, Tim Dahmen, and Philipp Slusallek. Interactive Headlight Simulation – A Case Study for Distributed Interactive Ray Tracing. In *Proceedings of Eurographics Workshop on Parallel Graphics and Visualization*, pages 81–88, 2002.

[28] Carsten Benthin, Ingo Wald, and Philipp Slusallek. A Scalable Approach to Interactive Global Illumination. *Computer Graphics Forum*, pages 621–630, 22 2003.

[29] Carsten Benthin, Ingo Wald, and Philipp Slusallek. Interactive Ray Tracing of Free-Form Surfaces. In *Proceedings of Afrigraph 2004*, pages 99–106, November 2004.

[30] Carsten Benthin, Ingo Wald, and Philipp Slusallek. Techniques for Interactive Ray Tracing of Bézier Surfaces. *Journal of Graphics Tools*, 11(2):1–16, 2006.

[31] Gary Bishop, Henry Fuchs, Leonard McMillan, and Ellen J. Scher Zagier. Frameless Rendering: Double Buffering Considered Harmful. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 175–176, 1994.

[32] Jiří Bittner, Michael Wimmer, Harald Piringer, and Werner Purgathofer. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum*, 23(3):615–624, 2004.

[33] Jiří Bittner and Peter Wonka. Visibility in Computer Graphics. Technical Report TR-186-2-03-03, Vienna University of Technology, 2003.

[34] Nicholas Blachford. Cell Architecture Explained Version 2. http://www.blachford.info/computer/Cell/Cell0_v2.html, 2005.

[35] James F. Blinn. Models of Light Reflection for Computer Synthesized Pictures. *Computer Graphics*, 11(2):192–198, 1977.

[36] James F. Blinn. Simulation of Wrinkled Surfaces. *Computer Graphics*, 12(3):286–292, 1978.

[37] James F. Blinn and Martin E. Newell. Texture and Reflection in Computer Generated Images. *Communications of the ACM*, 19(10):542–547, 1976.

[38] David Blythe. The Direct3D® 10 System. *ACM Transactions on Graphics*, 25(3):724–734, 2006.

[39] Ioana Boier-Martin, Holly Rushmeier, and Jingyi Jin. Parameterization of Triangle Meshes over Quadrilateral Domains. In *Proceedings of the Symposium on Geometry Processing*, pages 193–203, 2004.

[40] Mark R. Bolin and Gary W. Meyer. A Perceptually Based Adaptive Sampling Algorithm. *Computer Graphics*, 32(Annual Conference Series):299–309, 1998.

[41] Boost.org. Boost $\mu$Blas Sparse Vector. www.boost.org/libs/numeric/ublas/doc/vector_sparse.htm, 2002.

[42] Pavel Borodin, Stefan Gumhold, Michael Guthe, and Reinhard Klein. High-Quality Simplification with Generalized Pair Contractions. In *GraphiCon*, pages 147–154, September 2003.

[43] Pavel Borodin, Michael Guthe, and Reinhard Klein. Out-of-Core Simplification with Guaranteed Error Tolerance. In *Vision, Modeling and Visualisation*, pages 309–316, November 2003.

[44] Mario Botsch. *High Quality Surface Generation and Efficient Multiresolution Editing Based on Triangle Meshes*. PhD thesis, RWTH Aachen, 2005.

[45] Ronald Bracewell. *The Fourier Transform and Its Applications*. McGraw-Hill, NY, 3rd edition, 1999.

[46] Matthew Brand. Incremental Singular Value Decomposition of Uncertain Data with Missing Values. In *Proceedings of European Conference on Computer Vision*, pages 707–720, 2002.

[47] David Brayford, Martin Turner, and W. T. Hewitt. A Physical Based Spectral Model for Polarized Subsurface Light Scattering. In *Proceedings of Pacific Graphics*, pages 25–27, 2005.

[48] William Bricken. Virtual Reality: Directions of Growth. Technical Report R-90-1, University of Washington, Human Interface Technology Laboratory, 1990.

[49] I. N. Bronstein and K. A. Semendjajew. *Taschenbuch der Mathematik*, volume 20. Verlag Harri Deutsch, Thun und Frankfurt (Main), 1981.

[50] Frederick P. Brooks. What's Real About Virtual Reality? *IEEE Computer Graphics and Applications*, 19(6):16–27, November/December 1999.

[51] Michael R. Bussieck and Armin Pruessner. Mixed-Integer Nonlinear Programming. *SIAG/OPT Newsletter: Views & News*, 14(1):19–22, 2003.

[52] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional Reflection Functions from Surface Bump Maps. *Computer Graphics*, 21(4):273–281, 1987.

[53] Brian Cabral, Marc Olano, and Philip Nemec. Reflection Space Image Based Rendering. In *SIGGRAPH '99*, pages 165–170, Los Angeles, CA, 1999.

[54] Martin Čadík and Pavel Slavík. Evaluation of Two Principal Approaches to Objective Image Quality Assessment. In *Proceedings of the Eighth International Conference on Information Visualisation*, pages 513–518, 2004.

[55] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The Ray Engine. In *Proceedings of Graphics Hardware*, pages 37–46, September 2002.

[56] Nathan A. Carr, Jesse D. Hall, and John C. Hart. GPU Algorithms for Radiosity and Subsurface Scattering. In *Proceedings of Graphics Hardware*, pages 51–59, 2003.

[57] Christophe Cassagnabère, François Rousselle, and Christophe Renaud. Path Tracing using the AR350 Processor. In *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, 2004.

[58] Alan G. Chalmers, Ann McNamara, Scott Daly, Karol Myszkowski, and Tom Troscianko. Image Quality Metrics. SIGGRAPH Course 44, 2000.

[59] Bin Chan and Wenping Wang. Geocube – GPU accelerated real-time rendering of transparency and translucency. *The Visual Computer*, 21(8-10):579–590, 2005.

[60] Shenchang Eric Chen. QuickTime VR – An Image-Based Approach to Virtual Environment Navigation. *Computer Graphics*, 29(Annual Conference Series):29–38, 1995.

[61] Shenchang Eric Chen, Holly E. Rushmeier, Gavin Miller, and Douglass Turner. A Progressive Multi-Pass Method for Global Illumination. *ACM Computer Graphics*, 25(4):165–174, 1991.

[62] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, and Radek Grzeszczuk. Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. *ACM Transactions on Graphics*, 21(3):447–456, 2002.

[63] Yanyun Chen, Xin Tong, Jiaping Wang, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Shell Texture Functions. *ACM Transactions on Graphics*, 23(3):343–353, 2004.

[64] Jatin Chhugani and Subodh Kumar. View-dependent Adaptive Tessellation of Spline Surfaces. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 59–62, 2001.

[65] Jatin Chhugani and Subodh Kumar. Budget Sampling of Parametric Surface Patches. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 131–138, 2003.

[66] Martin Christen. Ray Tracing on GPU. Master's thesis, University of Applied Sciences Basel, Switzerland, 2005.

[67] Paolo Cignoni, Claudio Montani, Claudio Rocchini, Roberto Scopigno, and Marco Tarini. Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer*, 15(10):519–539, 1999.

[68] Paolo Cignoni, Claudio Rocchini, Claudio Montani, and Roberto Scopigno. External Memory Management and Simplification of Huge Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 9(4):525–537, 2003.

[69] Jonathan Cohen and Paul Debevec. gl.ict.usc.edu/HDRShop/lightgen/lightgen.html, 2001.

[70] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-Preserving Simplification. In *Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 115–122, 1998.

[71] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. Simplification Envelopes. In *Proc. of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 119–128, 1996.

[72] Michael F. Cohen and Donald P. Greenberg. The Hemi-Cube: A Radiosity Solution for Complex Environments. *Computer Graphics*, 19(3):31–40, 1985.

[73] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang Tiles for Image and Texture Generation. *ACM Transactions on Graphics*, 22(3):287–294, 2003.

[74] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational Shape Approximation. *ACM Transactions on Graphics*, 23(3):905–914, 2004.

[75] Gordon Collins and Adrian Hilton. Mesh Decimation for Displacement Mapping. In *Eurographics Short Papers*, 2002.

[76] Robert L. Cook. Shade Trees. *Computer Graphics*, 18(3):223–231, 1984.

[77] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. *Computer Graphics*, 18(3):137–145, 1984.

[78] Robert L. Cook and Kenneth E. Torrance. A Reflectance Model for Computer Graphics. *ACM Transactions on Graphics*, 1(1):7–24, 1982.

[79] Greg Coombe, Chad Hantak, Anselmo Lastra, and Radek Grzeszczuk. Online Construction of Surface Light Fields. In *Rendering Techniques*, pages 83–90, 2005.

[80] Greg Coombe, Mark J. Harris, and Anselmo Lastra. Radiosity on Graphics Hardware. In *Proc. of Graphics Interface*, pages 161–168, 2004.

[81] George Cross and Anil Jain. Markov Random Field Texture Models. *IEEE Transactions on Pattern and Machine Intelligence*, 5(1):25–39, 1983.

[82] Szabolcs Czuczor, László Szirmay-Kalos, László Szécsi, and László Neumann. Photon Map Gathering on the GPU. In *Eurographics Short Papers*, pages 117–120, 2005.

[83] Carsten Dachsbacher and Marc Stamminger. Reflective Shadow Maps. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 203–231, April 2005.

[84] Carsten Dachsbacher and Marc Stamminger. Splatting Indirect Illumination. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 93–100, 2006.

[85] Scott Daly. The Visible Difference Predictor: An algorithm for the assessment of image fidelity. In B. Watson, editor, *Digital Images and Human Vision*, pages 179–206. MIT Press, Cambridge, MA, USA, 1993.

[86] Cyrille Damez, Kirill Dmitriev, and Karol Myszkowski. State of the Art in Global Illumination for Interactive Applications and High-quality Animations. *Computer Graphics Forum*, 21(4):55–77, 2003.

[87] Kristin J. Dana, Bram van Ginneken, Shree K. Nayra, and Jan J. Koenderink. Reflectance and Texture of Real World Surfaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 151–157, San Juan, Puerto Rico, June 1997.

[88] Katja Daubert, Jan Kautz, Hans-Peter Seidel, Wolfgang Heidrich, and Jean-Michel Dischler. Efficient Light Transport Using Precomputed Visibility. *IEEE Comput. Graph. Appl.*, 23(3):28–37, 2003.

[89] Katja Daubert, Hendrik Lensch, Wolfgang Heidrich, and Hans-Peter Seidel. Efficient Cloth Modeling and Rendering. In *12th Eurographics Workshop on Rendering*, pages 63–70, 2001.

[90] Abhinav Dayal, Cliff Woolley, Benjamin Watson, and David Luebke. Adaptive Frameless Rendering. In *Rendering Techniques*, pages 265–275, 2005.

[91] Jeremy S. De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images. In *Proc. of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 361–368, 1997.

[92] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the Reflectance Field of a Human Face. In *Proc. of the 27th annual conference on Computer graphics and interactive techniques*, pages 145–156, 2000.

[93] Paul Debevec, Erik Reinhard, Sumanta Pattanaik, and Greg Ward. High-Dynamic-Range Imaging and Image-Based Lighting. SIGGRAPH Course 29, 2005.

[94] Paul Debevec, Andreas Wenger, Chris Tchou, Andrew Gardner, Jamie Waese, and Tim Hawkins. A Lighting Reproduction Approach to Live-Action Compositing. *ACM Transactions on Graphics*, 21(3):547–556, 2002.

[95] Paul E. Debevec and Jitendra Malik. Recovering High Dynamic Range Radiance Maps from Photographs. In *Proc. of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 369–378, 1997.

[96] Philippe Decaudin and Fabrice Neyret. Rendering Forest Scenes in Real-Time. In *Rendering Techniques*, pages 93–102, 2004.

[97] Xavier Décoret, Frédo Durand, François X. Sillion, and Julie Dorsey. Billboard Clouds for Extreme Model Simplification. *ACM Transactions on Graphics*, 22(2), 2003.

[98] Christopher DeCoro and Renato Pajarola. XFastMesh: Fast View-Dependent Meshing from External Memory. In *Proc. of Visualization 2002*, pages 363–370, Boston, Massachussetts, 2002.

[99] Christopher DeCoro and Szymon Rusinkiewicz. Pose-independent Simplification of Articulated Meshes. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 17–24, 2005.

[100] Patrick Degener, Jan Meseth, and Reinhard Klein. An Adaptable Surface Parameterization Method. In *12th International Meshing Roundtable*, pages 201–213, 2003.

[101] David E. DeMarle, Steven Parker, Mark Hartner, Christian Gribble, and Charles Hansen. Distributed Interactive Ray Tracing for Large Volume Visualization. In *Proc. of Symposium on Parallel and Large-Data Visualization and Graphics*, page 12, 2003.

[102] Kate Devlin, Alan Chalmers, Alexander Wilkie, and Werner Purgathofer. Tone Reproduction and Physically Based Spectral Rendering. In *Eurographics State of the Art Reports*, pages 101–123, Saarbrücken, Germany, 2002.

[103] Joel DeYoung and Alain Fournier. Properties of Tabulated Bidirectional Reflectance Distribution Functions. In *Proc. of the Conference on Graphics Interface*, pages 47–55, 1997.

[104] Pablo Diaz-Gutierrez, Meenakshisundaram Gopi, and Renato Pajarola. Hierarchyless Simplification, Stripification and Compression of Triangulated Two-Manifolds. *Computer Graphics Forum*, 24(3):457–467, 2005.

[105] Paul J. Diefenbach and Norman I. Badler. Multi-Pass Pipeline Rendering: Realism For Dynamic Environments. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 59–70, 1997.

[106] Andreas Dietrich, Ingo Wald, and Philipp Slusallek. Large-Scale CAD Model Visualization on a Scalable Shared-Memory Architecture. In *Proc. of Vision, Modeling, and Visualization*, pages 303–310, 2005.

[107] Mark A. Z. Dippé and Erling H. Wold. Antialiasing Through Stochastic Sampling. *Computer Graphics Forum*, 19(3):69–78, July 1985.

[108] Jean-Michel Dischler. Efficiently Rendering Macro Geometric Surface Structures with Bi-Directional Texture Functions. In *Rendering Techniques*, pages 169–180, 1998.

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

[110] Kirill Dmitriev, Thomas Annen, Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. A CAVE System for Interactive Modeling of Global Illumination in Car Interior. In *Proc. of Symposium on Virtual Reality Software and Technology*, pages 137–145, 2004.

[111] Kirill Dmitriev, Stefan Brabec, Karl Myszkowski, and Hana-Peter Seidel. Interactive Global Illumination Using Selective Photon Tracing. In *Eurographics Workshop on Rendering 2002*, pages 21–34, 2002.

[112] Junyu Dong and Mike J. Chantler. Comparison of Five 3D Surface Texture Synthesis Methods. In *Proc. of Texture Analysis and Synthesis*, pages 19–24, 2003.

[113] Craig Donner and Henrik Wann Jensen. Light diffusion in multi-layered translucent materials. *ACM Transactions on Graphics*, 24(3):1032–1039, 2005.

[114] Julie Dorsey and Holly Rushmeier. Digital Modeling of the Appearance of Materials. SIGGRAPH Course 26, 2005.

[115] Frédéric Drago and Karol Myszkowski. Validation Proposal for Global Illumination and Rendering Techniques. *Computers & Graphics*, 25(3):511–518, 2001.

[116] Reynald Dumont, Fabio Pellacini, and James A. Ferwerda. Perceptually-Driven Decision Theory for Interactive Realistic Rendering. *ACM Transactions on Graphics*, 22(2):152–181, 2003.

[117] Frédo Durand, Nicolas Holzschuch, Cyril Soler, Eric Chan, and François Sillion. A Frequency Analysis of Light Transport. *ACM Transactions on Graphics*, 24(3):1115–1126, 2005.

[118] Philip Dutré, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination*. A. K. Peters, 1st edition, July 2003.

[119] Philip Dutré, Henrik W. Jensen, Jim Arvo, Kavita Bala, Philippe Bekaert, Steve Marschner, and Matt Pharr. State of the Art in Monte Carlo Global Illumination. In *SIGGRAPH 2004 Courses*, August 2004.

[120] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proc. of SIGGRAPH 1995*, pages 173–182, 1995.

[121] Alexander Efremov, Vlastimil Havran, and Hans-Peter Seidel. Robust and Numerically Stable Bézier Clipping Method for Ray Tracing NURBS Surfaces. In *Proc. of the 21st Spring Conference on Computer Graphics*, pages 127–135, 2005.

[122] Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 341–346, 2001.

[123] Alexei A. Efros and Thomas K. Leung. Texture Synthesis by Non-Parametric Sampling. In *Proc. of the International Conference on Computer Vision – Volume 2*, pages 1033–1038, 1999.

[124] Jihad El-Sana, Elvir Azanli, and Amitabh Varshney. Skip Strips: Maintaining Triangle Strips for View-Dependent Rendering. In *Proc. of Visualization*, pages 131–138, 1999.

[125] Jihad El-Sana and Yi-Jen Chiang. External Memory View-Dependent Simplification. *Computer Graphics Forum*, 19(3):139–150, 2000.

[126] Jihad El-Sana and Amitabh Varshney. Generalized View-Dependent Simplification. *Computer Graphics Forum*, 18(3):83–94, 1999.

[127] David Eppstein and Meenakshisundaram Gopi. Single-Strip Triangulation of Manifolds with Arbitrary Topology. In *Proc. of the 20th Annual Symposium on Computational Geometry*, pages 455–456, 2004.

[128] Carl Erikson, Dinesh Manocha, and William Baxter. HLODs for Faster Display of Large Static and Dynamic Environments. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 111–120, 2001.

[129] Manfred Ernst, Tomas Akenine-Möller, and Henrik Wann Jensen. Interactive Rendering of Caustics using InterpolatedWarped Volumes. In *Proc. of Graphics Interface*, pages 87–96, 2005.

[130] Manfred Ernst, Christian Vogelgsang, and Günther Greiner. Stack Implementation on the Programmable Graphics Hardware. In *Proc. of Vision, Modeling, and Visualization 2004*, pages 255–262, 2004.

[131] Pau Estalella, Ignacio Martin, George Drettakis, and Dani Tost. A GPU-driven Algorithm for Accurate Interactive Reflections on Curved Objects. In *Rendering Techniques*, pages 313–318, 2006.

[132] Pau Estalella, Ignacio Martin, George Drettakis, Dani Tost, Olivier Devillers, and Frédéric Cazals. Accurate Interactive Specular Reflections on Curved Objects. In *Proc. of Vision, Modeling, and Visualization*, 2005.

[133] Fawaz Maamari et al. Test Cases to Assess the Accuracy of Lighting Computer Programs. Technical Report 171:2006, Commission International d'Éclairage, 2006.

[134] Mark D. Fairchild and Garrett M. Johnson. Meet iCAM: A Next-Generation Color Appearance Model. In *Proc. of IS&T/SID 10th Color Imaging Conference*, pages 33–38, 2002.

[135] Mark D. Fairchild and Garrett M. Johnson. Image Appearance Modeling. In *Proc. of IS&T/SPIE Electronic Imaging Conference*, pages 149–160, 2003.

[136] Guoliang Fan and Xiang-Gen Xia. Wavelet-based Texture Analysis and Synthesis Using Hidden Markov Models. *IEEE Transactions on Circuits and Systems, part I*, 50(1):106–120, January 2003.

[137] Hui Fang and John C. Hart. Textureshop: Texture Synthesis as a Photograph Editing Tool. *ACM Transactions on Graphics*, 23(3):354–359, 2004.

[138] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, 4th edition, 1996.

[139] Joshua Fender and Jonathan Rose. A High-Speed Ray Tracing Engine Built on a Field-Programmable System. In *Proc. of IEEE International Conference On Field-Programmable Technology*, pages 188–195, 2003.

[140] James A. Ferwerda. Elements of early vision for computer graphics. *IEEE Computer Graphics and Applications*, 21(5):22–33, 2001.

[141] James A. Ferwerda. Three Varieties of Realism in Computer Graphics. In *Proc. of SPIE*, pages 290–297, 2003.

[142] James A. Ferwerda, Sumanta Pattanaik, Peter Shirley, and Donald P. Greenberg. A Model of Visual Masking for Computer Graphics. *Computer Graphics*, 31:143–152, 1997.

[143] James A. Ferwerda and Fabio Pellacini. Functional Difference Predictors (FDPs): Measuring Meaningful Image Differences. In *Proc. of Asilomar Conference on Signals, Systems, and Computers*, pages 1388–1392, 2003.

[144] James A. Ferwerda, Stephen H. Westin, Randall C. Smith, and Richard Pawlicki. Effects of Rendering on Shape Perception in Automobile Design. In *Proc. of the 1st Symposium on Applied Perception in Graphics and Visualization*, pages 107–114, 2004.

[145] Jiri Filip and Michal Haindl. Non-linear Reflectance Model for Bidirectional Texture Function Synthesis. In *Proc. of the 17th International Conference on Pattern Recognition – Volume 1*, pages 80–83, 2004.

[146] George S. Fishman. *Monte-Carlo: Concepts, Algorithms, and Applications*. Springer-Verlag, New-York, 1996.

[147] Michael S. Floater and Kai Hormann. Surface Parameterization: a Tutorial and Survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 157–186. Springer Verlag, 2005.

[148] Leila De Floriani, Paolo Magillo, Enrico Puppo, and Davide Sobrero. A Multi-Resolution Topological Representation for Non-Manifold Meshes. In *Proc. of the 7th ACM Symposium on Solid Modeling and Applications*, Saarbrücken, Germany, 2002.

[149] Tim Foley and Jeremy Sugerman. KD-Tree Acceleration Structures for a GPU Raytracer. In *Proc. of Graphics Hardware*, pages 15–22, 2005.

[150] Arno Formella and Kerstin Müller. A Viewpoint-Dependent Approach to Ray Trace Free-Form Surfaces. *Computer Graphics Forum*, 23(2):143–156, 2004.

[151] David R. Forsey and R. Victor Klassen. An Adaptive Subdivision Algorithm for Crack Prevention in the Display of Parametric Surfaces. In *Proc. of Graphics Interface*, pages 1–8, May 1990.

[152] Alain Fournier. Normal Distribution Functions and Multiple Surfaces. In *Graphics Interface Workshop on Local Illumination*, pages 45–52, 1992.

[153] Alain Fournier. Separating Reflection Functions for Linear Radiosity. In *Rendering Techniques*, pages 296–305, 1995.

[154] Chi-Wing Fu and Man-Kang Leung. Texture Tiling on Arbitrary Topological Surfaces using Wang Tiles. In *Rendering Techniques*, pages 99–104, 2005.

[155] Ryo Furukawa, Hiroshi Kawasaki, Katsushi Ikeuchi, and Masao Sakauchi. Appearance based object modeling using texture database: Acquisition, compression and rendering. In *Rendering Techniques*, pages 257–266, June 2002.

[156] Michael Garland and Paul S. Heckbert. Surface Simplification Using Quadric Error Metrics. *Computer Graphics*, 31:209–216, 1997.

[157] Michael Garland and Paul S. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In David Ebert, Hans Hagen, and Holly Rushmeier, editors, *IEEE Visualization*, pages 263–270, 1998.

[158] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical Face Clustering on Polygonal Surfaces. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 49–58, 2001.

[159] Pascal Gautron, Jaroslav Krivánek, Sumatra Pattanaik, and Kadi Bouatouch. A Novel Hemispherical Basis for Accurate and Efficient Rendering. In *Rendering Techniques*, pages 321–330, 2004.

[160] Pascal Gautron, Jaroslav Křivánek, Kadi Bouatouch, and Sumanta N. Pattanaik. Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm. In *Rendering Techniques*, pages 55–64, June 2005.

[161] Markus Geimer and Oliver Abert. Interactive Ray Tracing of Trimmed Bicubic Bezier Surfaces without Triangulation. In *Proc. of the 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 71–78, 2005.

[162] Georgy L. Gimel'farb. *Image Textures and Gibbs Random Fields*. Springer-Verlag, New York, September 1999.

[163] Enrico Gobbetti and Fabio Marton. Far Voxels: A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms. *ACM Transactions on Graphics*, 24(3):878–885, August 2005.

[164] Michael Goesele. *New Acquisition Techniques for Real Objects and Light Sources in Computer Graphics*. Books on Demand, Norderstedt, Germany, December 2004.

[165] Michael Goesele, Hendrik P. A. Lensch, Jochen Lang, Christian Fuchs, and Hans-Peter Seidel. DISCO . Acquisition of Translucent Objects. *ACM Transactions on Graphics*, 23(3):835–844, 2004.

[166] Jay S. Gondek, Gary W. Meyer, and Jonathan G. Newman. Wavelength Dependent Reflectance Functions. In *Proc. of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 213–220, 1994.

[167] Meenakshisundaram Gopi and Dinesh Manocha. Simplifying Spline Models. *Computational Geometry: Theory and Applications*, 14(1-3):67–90, 1999.

[168] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the Interaction of Light between Diffuse Surfaces. *Computer Graphics*, 18(3):213–222, 1984.

[169] Steven Gortler, Radek Grzeszczuk, R. Szeliski, and Michael Cohen. The Lumigraph. In *Proc. of SIGGRAPH*, pages 43–54, 1996.

[170] Craig Gotsman, Stefan Gumhold, and Leif Kobbelt. Simplification and Compression of 3D Meshes. In *Tutorials on Multiresolution in Geometric Modelling*, pages 319–336. Springer-Verlag, Heidelberg, 2002.

[171] Xavier Granier and George Drettakis. Incremental Updates for Rapid Glossy Global Illumination. *Computer Graphisc Forum*, 20(3):268–277, 2001.

[172] Xavier Granier, George Drettakis, and Bruce Walter. Fast Global Illumination Including Specular Effects. In *Rendering Techniques*, pages 47–59, 2000.

[173] Xavier Granier and Wolfgang Heidrich. A Simple Layered RGB BRDF Model. *Graphics Models*, 65(4):171–184, 2003.

[174] Paul Green, Jan Kautz, Wojciech Matusik, and Frédo Durand. View-Dependent Precomputed Light Transport Using Nonlinear Gaussian Function Approximations. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 7–14, 2006.

[175] Donald P. Greenberg, Kenneth E. Torrance, Peter Shirley, James Arvo, James A. Ferwerda, Sumanta Pattanaik, Eric Lafortune, Bruce Walter, Sing-Choong Foo, and Ben Trumbore. A Framework for Realistic Image Synthesis. In *Proc. of SIGGRAPH*, pages 474–494, 1997.

[176] Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. The Irradiance Volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998.

[177] Jinwei Gu, Chien-I Tu, Ravi Ramamoorthi, Peter Belhumeur, Wojciech Matusik, and Shree Nayar. Time-Varying Surface Appearance: Acquisition, Modeling, and Rendering. *ACM Transactions on Graphics*, 25(3), 2006.

[178] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry Images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

[179] Johannes Günther, Tongbo Chen, Michael Goesele, Ingo Wald, and Hans-Peter Seidel. Efficient Acquisition and Realistic Rendering of Car Paint. In *Proc. of Vision, Modeling, and Visualization*, pages 487–494, 2005.

[180] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray Tracing Animated Scenes using Motion Decomposition. *Computer Graphics Forum*, 25(3):517–525, 2006.

[181] Johannes Günther, Ingo Wald, and Philipp Slusallek. Realtime Caustics using Distributed Photon Mapping. In *Rendering Techniques 2004*, pages 111–121, 2004.

[182] Thomas Günther, Christoph Poliwoda, Christof Reinhart, Jürgen Hesser, Reinhard Maenner, Hans-Peter Meinzer, and H.-J. Baur. VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine. In *Proc. of the 9th Eurographics Workshop on Graphics Hardware*, pages 103–108, September 1994.

[183] Michael Guthe, Ákos Balázs, and Reinhard Klein. Interactive High Quality Trimmed NURBS Visualization Using Appearance Preserving Tessellation. In *Proc. of TCVG Symposium on Visualization*, pages 211–220, 2004.

[184] Michael Guthe, Ákos Balázs, and Reinhard Klein. GPU-based trimming and tessellation of NURBS and T-Spline surfaces. *ACM Transactions on Graphics*, 24(3):1016–1023, August 2005.

[185] Michael Guthe, Pavel Borodin, Ákos Balázs, and Reinhard Klein. Real-Time Appearance Preserving Out-of-Core Rendering with Shadows. In *Rendering Techniques*, pages 69–79, 2004.

[186] Michael Guthe, Pavel Borodin, and Reinhard Klein. Efficient View-Dependent Out-of-Core Visualization. In *Proc. of the 4th International Conference on Virtual Reality and its Application in Industry*, pages 428–438, 2003.

[187] Michael Guthe, Pavel Borodin, and Reinhard Klein. Fast and accurate Hausdorff distance calculation between meshes. *Journal of WSCG*, 13(2):41–48, 2005.

[188] Michael Guthe and Reinhard Klein. Efficient NURBS Rendering using View-Dependent LOD and Normal Maps. *Journal of WSCG*, 11(1), February 2003.

[189] Michael Guthe, Jan Meseth, and Reinhard Klein. Fast and Memory Efficient View-Dependent trimmed NURBS Rendering. In *Proc. of Pacific Graphics*, pages 204–213, Oktober 2002.

[190] Stephane Guy and Cyril Soler. Graphics Gems Revisited. Fast and Physically-Based Rendering of Gemstones. *ACM Transactions on Graphics*, 23(3):231–238, 2004.

[191] Jörg Haber, Karol Myszkowski, Hitoshi Yamauchi, and Hans-Peter Seidel. Perceptually Guided Corrective Splatting. *Computer Graphics Forum*, 20(3):142–152, September 2001.

[192] Tom Haber, Tom Mertens, Philippe Bekaert, and Frank Van Reeth. A Computational Approach to Simulate Subsurface Light Diffusion in Arbitrarily Shaped Objects. In *Proc. of the 2005 Conference on Graphics Interface*, pages 79–86, 2005.

[193] Toshiya Hachisuka. High-Quality Global Illumination Rendering Using Rasterization. In *GPU Gems 2*, pages 615–634. NVidia, 2005.

[194] Paul Haeberli. Paint by Numbers: Abstract Image Representations. *Computer Graphics*, 25(4), August 1990.

[195] Michal Haindl, Jiri Grim, Petr Somol, Pavel Pudil, and Mineichi Kudo. A Multiscale Colour Texture Model. In *Proc. of the 17th International Conference on Pattern Recognition*, volume III, pages 177–180, 2004.

[196] Michal Haindl and Vojtech Havlíček. A simple multispectral multiresolution markov texture model. In M. Chantler, editor, *Proc. Texture 2002*, pages 63–66, 2002.

[197] Eric Haines and John Wallace. Shaft Culling for Efficient Ray-Traced Radiosity. In *Proc. of 2nd Eurographics Workshop on Rendering*, pages 225–234, 1999.

[198] Daniel Hall. The AR350: Today's ray trace rendering processor. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware - Hot 3D Presentations*, pages 13–19, 2001.

[199] David Halliday, Robert Resnick, and Jearl Walker. *Fundamentals of Physics*. John Wiley & Sons, 7th edition, 2004.

[200] Pat Hanrahan and Wolfgang Krueger. Reflection from Layered Surfaces due to Subsurface Scattering. In *Proc. of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 165–174, 1993.

[201] Xuejun Hao, Thomas Baby, and Amitabh Varshney. Interactive Subsurface Scattering for Translucent Meshes. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 75–82, 2003.

[202] Mark Harris, David Luebke, Ian Buck, Naga Govindaraju, Jens Krüger, Aaron E. Lefohn, Timothy J. Purcell, and Cliff Woolley. GPGPU: General-Purpose Computation on Graphics Hardware. SIGGRAPH Course Notes, Course 39, 2005.

[203] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[204] Miloš Hašan, Fabio Pellacini, and Kavita Bala. Direct-to-Indirect Transfer for Cinematic Relighting. *ACM Transactions on Graphics*, 25(3):1089–1097, 2006.

[205] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A Survey of Real-Time Soft Shadows Algorithms. In *Eurographics State of the Art Reports*, 2003.

[206] Michael Hauth, Olaf Etzmuss, Bernd Eberhardt, Reinhard Klein, Ralf Sarlette, Mirko Sattler, Katja Dauber, and Jan Kautz. Cloth Animation and Rendering. In *Eurographics 2002 Tutorials*, 2002.

[207] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University, Faculty of Electrical Engineering, Department of Computer Science and Engineering, 2000.

[208] Vlastimil Havran, Attila Neumann, Georg Zotti, Werner Purgathofer, and Hans-Peter Seidel. On Cross-Validation and Resampling of BRDF Data Measurements. In *Proc. of Spring Conference on Computer Graphics*, pages 161–168, 2005.

[209] Tim Hawkins, Jonathan Cohen, and Paul Debevec. A photometric approach to digitizing cultural artifacts. In *Proc. of Virtual Reality, Archeology, and Cultural Heritage*, pages 333–342, 2001.

[210] Taosong He, Lichan Hong, Arie Kaufman, Amitabh Varshney, and Sidney Wang. Voxel Based Object Simplification. In *Proc. of Visualization*, pages 296–303, 1995.

[211] Xiao D. He, Kenneth E. Torrance, François X. Sillion, and Donald P. Greenberg. A Comprehensive Physical Model for Light Reflection. *Computer Graphics*, 25(4):175–186, 1991.

[212] David J. Heeger and James R. Bergen. Pyramid-Based Texture Analysis/Synthesis. In *Proc. of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 229–238, 1995.

[213] Wolfgang Heidrich. Interactive Display of Global Illumination Solutions for Non-diffuse Environments - A Survey. *Computer Graphics Forum*, 20(4):225–243, 2001.

[214] Wolfgang Heidrich, Katja Daubert, Jan Kautz, and Hans-Peter Seidel. Illuminating Micro Geometry Based on Precomputed Visibility. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 455–464, 2000.

[215] Wolfgang Heidrich, Hendrik Lensch, Michael Cohen, and Hans-Peter Seidel. Light Field Techniques for Reflections and Refractions. In *Rendering Techniques*, pages 187–196, 1999.

[216] Wolfgang Heidrich and Hans-Peter Seidel. View-Independent Environment Maps. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware '98*, pages 39–45, 1998.

[217] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, Hardware-Accelerated Shading and Lighting. In *SIGGRAPH '99*, pages 171–178, Los Angeles, CA, 1999.

[218] Justin Hensley, Thorsten Scheuermann, Greg Coombe, Montek Singh, and Anselmo Lastra. Fast Summed-Area Table Generation and its Applications. *Computer Graphics Forum*, 24(3):547–555, 2005.

[219] Karl E. Hillesland, Sergey Molinov, and Radek Grzeszczuk. Nonlinear Optimization Framework for Image-Based Modeling on Programmable Graphics Hardware. *ACM Transactions on Graphics*, 22(3):925–934, 2003.

[220] Hideki Hirayama, Kazufumi Kaneda, Hideo Yamashita, and Yoshimi Monden. An Accurate Illumination Model for Objects Coated with Multilayer Films. In *Eurographics Short Presentations*, pages 145–150, 2000.

[221] Hugues Hoppe. Progressive Meshes. *Computer Graphics*, 30(Annual Conference Series):99–108, 1996.

[222] Hugues Hoppe. View-Dependent Refinement of Progressive Meshes. *Computer Graphics*, 31(Annual Conference Series):189–198, 1997.

[223] Hugues Hoppe. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 59–66, San Francisco, 1999.

[224] Hugues Hoppe. Optimization of Mesh Locality for Transparent Vertex Caching. In *Proc. of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 269–276, 1999.

[225] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh Optimization. In *Proc. of the 20th Annual conference on Computer Graphics and Interactive Techniques*, pages 19–26, 1993.

[226] Xianyou Hou, Li-Yi Wei, Heung-Yeung Shum, and Baining Guo. Real-rime Multi-perspective Rendering on Graphics Hardware. In *Rendering Techniques*, pages 93–102, 2006.

[227] P. Hough. Method and Means for Recognizing Complex Patterns. US patent 3,069,654, 1962.

[228] Greg Humphreys and C. Scott Ananian. TigerSHARK: A Hardware Accelerated Ray-tracing Engine. Technical report, Princeton University, 1996.

[229] Milo Hyben, Ilja martisovits, and Andrej Ferko. Scene Complexity for Rendering in Flatland. In *Proc. of the Spring Conference on Computer Graphics*, pages 112–120, 1998.

[230] Isabelle Icart and Didier Arquès. A Physically-Based BRDF Model for Multilayer Systems with Uncorrelated Rough Boundaries. In *Rendering Techniques*, pages 353–364, 2000.

[231] Industrial Light & Magic. OpenEXR website. www.openexr.com, 2006.

[232] Martin Isenburg, Peter Lindstrom, Stefan Gumhold, and Jack Snoeyink. Large Mesh Simplification using Processing Sequences. In *Proc. of Visualization*, pages 465–472, 2003.

[233] Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. An Efficient Method for Rendering Underwater Optical Effects Using Graphics Hardware. *Computer Graphics Forum*, 21(4):701–701, 2002.

[234] Kei Iwasaki, Fujiichi Yoshimoto, Yoshinori Dobashi, and Tomoyuki Nishita. A Rapid Rendering Method for Caustics Arising from Refraction by Transparent Objects. In *Proc. of International Conference on Cyberworlds*, pages 39–44, 2004.

[235] Kei Iwasaki, Fujiichi Yoshimoto, Yoshinori Dobashi, and Tomoyuki Nishita. A Fast Rendering Technique of Transparent Objects and Caustics. In *Proc. of International Conference on Computer Animation and Social Agents*, pages 165–170, 2005.

[236] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics*, 22(3):879–887, 2003.

[237] Frederik W. Jansen and Jarke J. van Wijk. Fast Previewing Techniques in Raster Graphics. In *Proceeding of Eurographics*, pages 195–202, 1983.

[238] Henrik W. Jensen. Importance Driven Path Tracing using Photon Maps. In *Proc. of Eurographics Rendering Workshop*, pages 326–335, 1995.

[239] Henrik W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, 2001.

[240] Henrik W. Jensen and Juan Buhler. A Rapid Hierarchical Rendering Technique for Translucent Materials. *ACM Transactions on Graphics*, 21(3):576–581, 2002.

[241] Henrik W. Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A Practical Model for Subsurface Light Transport. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 511–518, 2001.

[242] Stefan Jeschke, Michael Wimmer, and Werner Purgathofer. Image-based Representations for Accelerated Rendering of Complex Scenes. In *Eurographics State of the Art Reports*, pages 1–20, August 2005.

[243] Juan-Roberto Jiménez, Karol Myszkowski, and Xavier Pueyo. Interactive Global Illumination in Dynamic Participating Media Using Selective Photon Tracing. In *Proc. of the 21st Spring Conference on Computer graphics*, pages 211–218, 2005.

[244] Jingyi Yu and Leonard McMillan. A Framework for Multiperspective Rendering. In *Eurographics Symposium on Rendering*, pages 61–68, Norrköping, Sweden, 2004.

[245] Jingyi Yu and Leonard McMillan. Modelling Reflections via Multiperspective Imaging. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 117–124, San Diego, CA, 2005.

[246] Joint Photographic Experts Group. *ISO/IEC IS 10918-1*, August 1990.

[247] Bela Julesz. Visual Pattern Discrimination. *IRE Transactions on Information Theory*, 8(2):84–92, 1962.

[248] Fehrenc Kahlesz, Ákos Balázs, and Reinhard Klein. Multiresolution Rendering By Sewing Trimmed NURBS Surfaces. In *7th ACM Symposium on Solid Modeling and Applications 2002*, pages 281–288, 2002.

[249] James T. Kajiya. The Rendering Equation. *Computer Graphics*, 20(4):143–150, 1986.

[250] A. D. Kalvin and R. H. Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. *IEEE Computer Graphics and Applications*, 16(3):65–77, 1997.

[251] Nanda Kambhatla and Todd Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9:1493–1516, 1997.

[252] Tomomichi Kaneko, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda, and Susumu Tachi. Detailed Shape Representation with Parallax Mapping. In *Proc. of the 11th International Conference on Artificial Reality and Telexistence*, pages 205–208, 2001.

[253] Filip Karlsson and Carl Johan Ljungstedt. Ray tracing fully implemented on programmable graphics hardware. Master's thesis, Chalmers University of Technology, Göteborg, Sweden, 2004.

[254] K. F. Karner and M. Prantl. A Concept for Evaluating the Accuracy of Computer-Generated Images. In *Proc. of the Twelfth Spring Conference on Computer Graphics*, pages 145–154, 1996.

[255] Jan Kautz. Hardware Lighting and Shading: a Survey. *Computer Graphics Forum*, 23(1):85–112, 2004.

[256] Jan Kautz, Katja Dauber, and Hans-Peter Seidel. Advanced environment mapping in VR applications. *Computers & Graphics*, 28(1):99–104, 2003.

[257] Jan Kautz, Jaakko Lehtinen, and Timo Aila. Hemispherical Rasterization for Self-Shadowing of Dynamic Objects. In *Proc. of Eurographics Symposium on Rendering 2004*, pages 179–184, 2004.

[258] Jan Kautz and Michael McCool. Interactive Rendering with Arbitrary BRDFs using Separable Approximations. In *Tenth Eurographics Workshop on Rendering*, pages 281–292, 1999.

[259] Jan Kautz and Michael McCool. Approximation of Glossy Reflection with Prefiltered Environment Maps. In *Proc. of Graphics Interface 2000*, pages 119–126, 2000.

[260] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonic. In *13th Eurographics Workshop on Rendering*, pages 301–308, 2002.

[261] Jan Kautz, Pere-Pau Vázquez, Wolfgang Heidrich, and Hans-Peter Seidel. A Unified Approach to Prefiltered Environment Maps. In *11th Eurographics Workshop on Rendering*, pages 185–196, 2000.

[262] Tomoyuki Nishita Kei Iwasaki, Yoshinori Dobashi. A Fast Rendering Method for Refractive and Reflective Caustics Due to Water Surfaces. *Computer Graphics Forum*, 22(3):601–610, 2003.

[263] Csaba Kelemen and László Szirmay-Kalos. A Microfacet Based Coupled Specular-Matte BRDF Model with Importance Sampling. In *Eurographics Short Presentations*, pages 25–34, 2001.

[264] Alexander Keller. Instant Radiosity. In *Proc. of SIGGRAPH*, pages 49–56, 1997.

[265] Alexander Keller, Thomas Kollig, Mateu Sbert, and László Szirmay-Kalos. Efficient Monte Carlo and Quasi-Monte Carlo Rendering Techniques. In *Eurographics Tutorials*, 2003.

[266] Youngihn Kho and Michael Garland. User-Guided Simplification. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 123–126, 2003.

[267] Junho Kim and Seungyong Lee. Truly Selective Refinement of Progressive Meshes. In *Proc. of Graphics Interface*, pages 101–110, 2001.

[268] Scott Kircher and Michael Garland. Progressive Multiresolution Meshes for Deforming Surfaces. In *Proc. of Symposium on Computer Animation*, pages 191–200, 2005.

[269] Masaki Kitahara, Hideaki Kimata, Kazuto Kamikura, and Yoshiyuki Yashima. Progressive Compression of Surface Light Fields. *NTT Technical Review*, 2(8):27–34, August 2004.

[270] Reinhard Klein. Multiresolution Representations for Surfaces Meshes based on the Vertex Decimation Method. *Computer and Graphics*, 22(1):13–26, 1998.

[271] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. Mesh Reduction with Error Control. In *IEEE Visualization '96*, pages 311–318, 1996.

[272] Reinhard Klein, Jan Meseth, Gero Müller, Ralf Sarlette, Michael Guthe, and Ákos Balázs. RealReflect: Real-time Visualization of Complex Reflectance Behaviour in Virtual Prototyping. In *Eurographics Industrial and Project Presentations*, 2003.

[273] Reinhard Klein and Wolfgang Strasser. Large Mesh Generation from Boundary Models with Parametric Face Representation. In *Proc. of Symposium on Solid Modeling*, pages 431–440, 1995.

[274] Günter Knittel and Wolfgang Straßer. VIZARD – Visualization Accelerator for Realtime Display. In *Proc. of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 139–146, 1997.

[275] Jan J. Koenderink, Andrea J. van Doorn, and Marigo Stavridi. Bidirectional Reflection Distribution Function Expressed in Terms of Surface Scattering Modes. In *Proc. of the 4th European Conference on Computer Vision – Volume II*, pages 28–39, 1996.

[276] Thomas Kollig and Alexander Keller. Efficient Illumination by High Dynamic Range Images. In *Proc. of the 14th Eurographics Workshop on Rendering*, pages 45–50, 2003.

[277] Janne Kontkanen, Emmanuel Turquin, Nicolas Holzschuch, and François Sillion. Wavelet Radiance Transport for Interactive Indirect Lighting. In *Rendering Techniques*, pages 161–171, 2006.

[278] Melissa L. Koudelka, Sebastian Magda, Peter N. Belhumeur, and David L. Kriegman. Acquisition, Compression, and Synthesis of Bidirectional Texture Functions. In *Proc. of 3rd International Workshop on Texture Analysis and Synthesis*, pages 59–64, 2003.

[279] Vít Kovalĉik and Jiří Sochor. Occlusion Culling with Statistically Optimized Occlusion Queries. In *WSCG (Short Papers)*, pages 109–112, 2005.

[280] Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Lightness Perception in Tone Reproduction for High Dynamic Range Images. *Computer Graphics Forum*, 24(3):635–645, 2005.

[281] Wolfram Kresse, Dirk Reiners, and Christian Knöpfle. Color Consistency for Digital Multi-Projector Stereo Display Systems: The HEyeWall and The Digital CAVE. In *Proc. of the Workshop on Virtual Environments*, pages 271–279, 2003.

[282] Anders W. Kristensen, Tomas Akenine-Möller, and Henrik W. Jensen. Precomputed Local Radiance Transfer for Real-Time Lighting Design. *ACM Transactions on Graphics*, 24(3):1208–1215, 2005.

[283] Jens Krüger and Rüdiger Westermann. Interactive Screen-Space Accurate Photon Tracing on GPUs. In *Rendering Techniques*, pages 319–329, 2006.

[284] Bob Kuehne, Tom True, Alan Commike, and Dave Shreiner. Performance OpenGL: Platform Independent Techniques. SIGGRAPH Course #19, 2005.

[285] Subodh Kumar, Dinesh Manocha, and Anselmo Lastra. Interactive display of large-scale NURBS models. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):323–336, 1996.

[286] Subodh Kumar, Dinesh Manocha, Hansong Zhang, and Kenneth E. Hoff. Accelerated Walkthrough of Large Spline Models. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 91–102. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

[287] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture Optimization for Example-Based Synthesis. *ACM Transactions on Graphics*, 24(3):795–802, 2005.

[288] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[289] Eric P. Lafortune and Yves D. Willems. A Theoretical Framework for Physically based Rendering. *Computer Graphics Forum*, 13(2):97–107, 1994.

[290] Eric P. F. Lafortune, Sing-Choong Foo, Kenneth E. Torrance, and Donald P. Greenberg. Nonlinear Approximation of Reflectance Functions. In *Proc. of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 117–126, 1997.

[291] Paul Lalonde and Alain Fournier. A Wavelet Representation of Reflectance Functions. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):329–336, 1997.

[292] Hayden Landis. Production-Ready Global Illumination. SIGGRAPH Course Notes #16, 2002.

[293] Bent D. Larsen and Niels J. Christensen. Simulating Photon Mapping for Real-time Applications. In *Proc. of 15th Eurographics Symposium on Rendering*, pages 123–131, 2004.

[294] Lutz Latta and Andreas Kolb. Homomorphic Factorization of BRDF-based Lighting Computation. *ACM Transactions on Graphics*, 21(3):509–516, 2002.

[295] Jason Lawrence, Aner Ben-Artzi, Christopher DeCoro, Wojciech Matusik, Hanspeter Pfister, Ravi Ramamoorthi, and Szymon Rusinkiewicz. Inverse Shade Trees for Non-Parametric Material Representation and Editing. *ACM Transactions on Graphics*, 25(3):735–745, 2006.

[296] Laurent Lefebvre and Pierre Poulin. Extraction and Synthesis of Bump Maps from Photograph. In *Graphics Interface Posters Proceedings*, pages 12–13, 2000.

[297] Sylvain Lefebvre and Hugues Hoppe. Parallel Controllable Texture Synthesis. *ACM Transactions on Graphics*, 24(3):777–786, 2005.

[298] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Octree Textures on the GPU. In *GPU Gems 2*, pages 595–613. NVidia, 2005.

[299] Aaron Lefohn, Joe M. Kniss, Robert Strzodka, Shubhabrata Sengupta, and John D. Owens. Glift: Generic, Efficient, Random-Access GPU Data Structures. *ACM Transactions on Graphics*, 25(1):60–99, 2006.

[300] Jaakko Lehtinen and Jan Kautz. Matrix Radiance Transfer. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 59–64, April 2003.

[301] Hendrik P. Lensch, Michael Goesele, Philippe Bekaert, Jan Kautz, Marcus A. Magnor, Jochen Lang, and Hans-Peter Seidel. Interactive Rendering of Translucent Objects. In *Proc. of the 10th Pacific Conference on Computer Graphics and Applications*, pages 214–224, 2002.

[302] Hendrik P. Lensch, Michael Gösele, Yung-Yu Chuang, Tim Hawkins, Steve Marschner, Wojciech Matusik, and Gero Müller. Realistic Materials in Computer Graphics. In SIGGRAPH 2005 Tutorials, 2005.

[303] Hendrik P. Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. Image-Based Reconstruction of Spatial Appearance and Geometric Detail. *ACM Transactions on Graphics*, 22(2):234–257, 2003.

[304] Thomas Leung and Jitendra Malik. Representing and Recognizing the Visual Appearance of Materials using Three-dimensional Textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[305] Michael Levoy and Pat Hanrahan. Light Field Rendering. In *Proc. of SIGGRAPH 1996*, pages 31–42, 1996.

[306] Bruno Levy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least Squares Conformal Mappings for Automatic Texture Atlas Generation. *ACM Transactions on Graphics*, 21(3):363–371, 2002.

[307] Robert R. Lewis. Making Shaders More Physically Plausible. *Computer Graphics Forum*, 13(2):109–120, 1994.

[308] Jonas Lext and Tomas Akenine-Moeller. Towards Rapid Reconstruction for Animated Ray Tracing. In *Eurographics Short Papers*, pages 311–318, 2001.

[309] Bei Li, Gary W. Meyer, and R. Victor Klassen. A Comparison of Two Image Quality Models. In *Proc. of SPIE*, pages 98–109, 1998.

[310] Stan Z. Li. *Markov Random Field Modeling in Computer Vision*. Springer-Verlag, London, UK, 1995.

[311] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-Time Texture Synthesis by Patch-Based Sampling. *ACM Transactions on Graphics*, 20(3):127–150, 2001.

[312] Wen-Chieh Lin, James H. Hays, Chenyu Wu, Vivek Kwatra, and Yanxi Liu. A Comparison Study of Four Texture Synthesis Algorithms on Regular and Near-regular Textures. Technical Report CMU-RI-TR-04-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2004.

[313] Erik Lindholm, Mark J. Kligard, and Henry Moreton. A User-Programmable Vertex Engine. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 149–158, 2001.

[314] Cliff Lindsay and Emmanuel Agu. Spherical Harmonic Lighting of Wavelength-Dependent Phenomena. In *Eurographics Short Papers*, pages 121–124, 2005.

[315] Peter Lindstrom. Out-of-core construction and visualization of multiresolution surfaces. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 93–102, 2003.

[316] Peter Lindstrom and Greg Turk. Fast and Memory Efficient Polygonal Simplification. In *Proc. of Visualization*, pages 279–286, 1998.

[317] Peter Lindstrom and Greg Turk. Image-Driven Simplification. *ACM Transactions on Graphics*, 19(3):204–241, 2000.

[318] Jun S. Liu. *Monte-Carlo Strategies for Scientific Computing*. Springer-Verlag, New-York, 2001.

[319] Xinguo Liu, Yaohua Hu, Jingdan Zhang, Xin Tong, Baining Guo, and Heung-Yeung Shum. Synthesis and Rendering of Bidirectional Texture Functions on Arbitrary Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(3):278–289, 2004.

[320] Xinguo Liu, Peter-Pike Sloan, Heung-Yeung Shum, and John Snyder. All-Frequency Precomputed Radiance Transfer for Glossy Objects. In *Rendering Techniques*, pages 337–344, 2004.

[321] Xinguo Liu, Yizhou Yu, and Heung-Yeung Shum. Synthesizing Bidirectional Texture Functions for Real-World Surfaces. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 97–106, 2001.

[322] Yanxi Liu, Robert T. Collins, and Yanghai Tsin. A Computational Model for Periodic Perception Based on Frieze and Wallpaper Goups. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):354–371, March 2004.

[323] Yanxi Liu, Wen-Chieh Lin, and James H. Hays. Near Regular Texture Analysis and Manipulation. *ACM Transactions on Graphics*, 23(3):368–376, 2004.

[324] Yanxi Liu, Yanghai Tsin, and Wen-Chieh Lin. The Promise and Perils of Near-Regular Texture. *International Journal of Computer Vision*, 62(1–2):145–159, April 2005.

[325] Stuart P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[326] Peter Longhurst and Alan Chalmers. User Validation of Image Quality Assessment Algorithms. In *EGUK 04, Theory and Practice of Computer Graphics*, pages 196–202, June 2004.

[327] Peter Longhurst, Patrick Ledda, and Alan Chalmers. Psychophysically based Artistic Techniques for Increased Perceived Realism of Virtual Environments. In *Proc. of the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pages 123–132, 2003.

[328] Jeffrey Lubin. A Visual Discrimination Model for Imaging System Design and Development. In E. Peli, editor, *Vision Models for Target Detection and Recognition*, volume 2, pages 245–283. World Scientific, 1995.

[329] David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, San Francisco, 1st edition, 2002.

[330] Wan-Chun Ma, Sung-Hsiang Chao, Yu-Ting Tseng, Yung-Yu Chuang adn Chun-Fa Chang, Bing-Yu Chen, and Ming Ouhyoung. Level-of-Detail Representation of Bidirectional Texture Functions for Real-Time Rendering. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 187–194, 2005.

[331] Fawaz Maamari and Marc Fontoynont. Analytical Tests for Investigating the Accuracy of Lighting Programs. *Lighting Research and Technology*, 35(3):225–242, 2003.

[332] Paulo Maciel and Peter Shirley. Visual Navigation of Large Environments using Textured Clusters. In *Proc. of the Symposium on Interactive 3D Grapics*, pages 95–102, 1995.

[333] Thomas M. MacRobert and Ian N. Sneddon. *Spherical Harmonics: An Elementary Treatise on Harmonic Functions, with Applications*. Pergamon Press, Oxford, England, 3rd ed. rev. edition, 1967.

[334] Sebastian Magda and David L. Kriegman. Fast Texture Synthesis on Arbitrary Meshes. In *Rendering Techniques*, pages 82–89, June 2003.

[335] Nadia Magnenat-Thalmann, Frederic Cordier, Pascal Volino, Michael Keckeisen, Stefan Kimmerle, Reinhard Klein, and Jan Meseth. Simulation of Clothes for Real-time Applications. Eurographics 2004, Tutorial 1: Simulation of Clothes for Real-time Applications, 2004.

[336] Nihar R. Mahapatra and Balakrishna Venkatrao. The Processor-Memory bottleneck: Problems and Solutions. *Crossroads*, 5(3es):2, 1999.

[337] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, and Mark Horowitz. Smart Memories: A Modular Reconfigurable Architecture. In *Proc. of the 27th Annual International Symposium on Computer Architecture*, pages 161–171, 2000.

[338] Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial Texture Maps. In *Proc. of the 28th annual conference on Computer graphics and interactive techniques*, pages 519–528, 2001.

[339] Carol Manetta and Richard A. Blade. Glossary of Virtual Reality Terminology. *International Journal of Virtual Reality*, 1(2):35–39, 1995.

[340] RafaŁl Mantiuk, Karol Myszkowski, and Hans-Peter Seidel. Visible Difference Predicator for High Dynamic Range Images. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, pages 2763–2769, 2004.

[341] John Mardaljevic. *Daylight Simulation: Validation, Sky Models and Daylight Coefficients*. PhD thesis, De Montfort University, Leicester, Institute of Energy and Sustainable Development, 1999.

[342] Martin Marinov and Leif Kobbelt. Automatic Generation of Structure Preserving Multiresolution Models. *Computer Graphics Forum*, 24(3):479–486, 2005.

[343] William L. Martens and Karol Myszkowski. Psychophysical Validation of the Visible Differences Predictor for Global Illumination Applications. In *Proc. of IEEE Visualization, Late Breaking Hot Topics*, 1998.

[344] Vincent Masselus. *A Practical Framework for Fixed Viewpoint Image-Based Relighting*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 2005.

[345] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A Data-Driven Reflectance Model. *ACM Transactions on Graphics*, 22(3):759–769, 2003.

[346] Nelson Max. Shadows for bump mapped surfaces. In T. L. Kunii, editor, *Advanced Computer Graphics*, pages 145–156. Springer Verlag, Tokyo, 1986.

[347] David K. McAllister. *A Generalized Surface Appearance Representation for Computer Graphics*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 2002.

[348] David K. McAllister, Anselmo Lastra, and Wolfgang Heidrich. Efficient Rendering of Spatial Bi-directional Reflectance Distribution Functions. In *Proc. of Graphics Hardware 2002*, pages 79–88, Saarbrucken, Germany, 2002. Eurographics Association. ISBN:1-58113-580-7.

[349] Michael D. McCool, Jason Ang, and Anis Ahmad. Homomorphic factorization of brdfs for high-performance rendering. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 171–178, 2001.

[350] Leonard McMillan and Gary Bishop. Head-tracked stereoscopic display using image warping. In *Proc. of SPIE – Volume 2409*, pages 21–30, 1995.

[351] Ann McNamara. Visual Perception in Realistic Image Synthesis. *Computer Graphics Forum*, 20(4):221–224, 2001.

[352] Ann McNamara, Alan Chalmers, Tom Troscianko, and Iain Gilchrist. Comparing Real and Synthetic Scenes using Human Judgements of Lightness. In *Rendering Techniques*, pages 207–218, 2000.

[353] Chunhui Mei, Jiaoying Shi, and Fuli Wu. Rendering with Spherical Radiance Transport Maps. *Computer Graphics Forum*, 23(3):281–290, 2004.

[354] Michael Meißner, Urs Kanus, and Wolfgang Straßer. VIZCARD II, A PCI-Card for Real-Time Volume Rendering. In *Proc. of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 61–67, 1998.

[355] Tom Mertens, Jan Kautz, Philippe Bekaert, Hans-Peter Seidel, and Frank Van Reeth. Interactive Rendering of Translucent Deformable Objects. In *Rendering Techniques*, pages 130–140, 2003.

[356] Jan Meseth, Michael Guthe, and Reinhard Klein. Interactive Fragment Tracing. *The Visual Computer*, 21(8–11):591–600, 2005.

[357] Jan Meseth and Reinhard Klein. Memory Efficient Billboard Clouds for BTF Textured Objects. In *Proc. of Vision, Modeling, and Visualization*, pages 167–174, 2004.

[358] Jan Meseth, Gero Müller, and Reinhard Klein. Preserving realism in real-time rendering of bidirectional texture functions. In *OpenSG Symposium 2003*, pages 89–96. Eurographics Association, Switzerland, April 2003.

[359] Jan Meseth, Gero Müller, and Reinhard Klein. Reflectance Field based real-time, high-quality Rendering of Bidirectional Texture Functions. *Computers and Graphics*, 28(1):103–112, February 2004.

[360] Jan Meseth, Gero Müller, Reinhard Klein, Florian Röder, and Michael Arnold. Verification of Rendering Quality from Measured BTFs. In *Proc. of Symposium on Applied Perception in Graphics and Visualization*, pages 127–134, 2006.

[361] Jan Meseth, Gero Müller, Mirko Sattler, and Reinhard Klein. BTF Rendering for Virtual Environments. In *Virtual Concepts 2003*, pages 356–363, November 2003.

[362] Gary W. Meyer, Holly E. Rushmeier, Michael F. Cohen, Donald P. Greenberg, and Kenneth E. Torrance. An Experimental Evaluation of Computer Graphics Imagery. *ACM Transactions on Graphics*, 5(1):30–50, 1986.

[363] Gavin Miller, Steven Rubin, and Dulce Ponceleon. Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. In *9th Eurographics Workshop on Rendering*, pages 281–292, 1998.

[364] Gene Miller and Robert Hoffman. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. Advanced Image Synthesis Course Notes, Siggraph Conference, 1984.

[365] M. Minnaert. The Reciprocity Principle in Lunar Photometry. *Astrophysical Journal*, 93:403–410, 1941.

[366] Alex Mohr and Michael Gleicher. Deformation Sensitive Decimation. Technical report, University of Wisconsin, 2003.

[367] David Mount and Sunil Arya. ANN: A Library for Approximate Nearest Neighbor Searching. www.cs.umd.edu/∼mount/ANN/, 2005.

[368] Gero Müller, Gerhard H. Bendels, and Reinhard Klein. Rapid Synchronous Acquisition of Geometry and BTF for Cultural Heritage Artefacts. In *Proc. of the 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage*, pages 13–20, November 2005.

[369] Gero Müller, Jan Meseth, and Reinhard Klein. Compression and real-time Rendering of Measured BTFs using local PCA. In *Proc. of Vision, Modeling and Visualisation*, pages 271–280, November 2003.

[370] Gero Müller, Jan Meseth, and Reinhard Klein. Fast Environmental Lighting for Local-PCA Encoded BTFs. In *Computer Graphics International*. IEEE Computer Society Press, June 2004.

[371] Gero Müller, Jan Meseth, Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Acquisition, Synthesis and Rendering of Bidirectional Texture Functions. *Computer Graphics Forum*, 24(1):83–109, 2005.

[372] Gero Müller, Ralf Sarlette, and Reinhard Klein. Data-driven Local Coordinate Systems for Image-Based Rendering. *Computer Graphics Forum*, 25(3):369–378, 2006.

[373] Michael J. Muuss. Towards Real-Time Ray-Tracing of Combinatorial Solid Geometric Objects. In *Proc. of BRL-CAD Symposium*, June 1995.

[374] Karol Myszkowski, Wolfgang Heidrich, Michael Goesele, Bernd Höfflinger, Grzegorz Krawczyk, and Matthew Trentacoste. High Dynamic Range Techniques in Graphics: from Acquisition to Display. Eurographics Tutorials, 2005.

[375] Shree K. Nayar, Katsushi Ikeuchi, and Takeo Kanade. Surface Reflection: Physical and Geometrical Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):611–634, 1991.

[376] Andrew Nealen and Marc Alexa. Hybrid Texture Synthesis. In *Rendering Techniques*, pages 97–105, 2003.

[377] A. Nemcsics. The Color Space of the Coloroid Color Order System. *Color Research and Application*, 12:135–146, 1987.

[378] László Neumann, Kresimir Matkovic, and Werner Purgathofer. Perception Based Color Image Difference. *Computer Graphics Forum*, 17(3):233–242, 1998.

[379] László Neumann and Attila Neumann. Photosimulation: Interreflection with Arbitrary Reflectance Models and Illumination. *Computer Graphics Forum*, 8(1):21–34, 1989.

[380] László Neumann, Werner Purgathofer, Robert F. Tobler, Attila Neumann, Pavol Elias, Martin Feda, and Xavier Pueyo. The Stochastic Ray Method for Radiosity. In *Rendering Techniques*, pages 206–218, 1995.

[381] Fabrice Neyret and Marie-Paule Cani. Pattern-Based Texturing Revisited. In *Proc. of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 235–242, 1999.

[382] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-Frequency Shadows using Non-Linear Wavelet Lighting Approximation. *ACM Transactions on Graphics*, 22(3):376–381, 2003.

[383] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. Triple Product Wavelet Integrals for All-Frequency Relighting. *ACM Transactions on Graphics*, 23(3):477–487, 2004.

[384] Addy Ngan, Frédo Durand, and Wojciech Matusik. Experimental Analysis of BRDF Models. In *Rendering Techniques*, pages 117–126, 2006.

[385] Addy Ngan, Frédo Durand, and Wojciech Matusik. Image-driven Navigation of Analytical BRDF Models. *Rendering Techniques*, pages 399–407, 2006.

[386] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Geometrical Considerations. and Nomenclature for Reflectance. Technical Report BMS Monograph 160, National Bureau of Standards, 1977.

[387] André Nicoll. Synthese regelmäßiger Texturen. Master's thesis, Rheinische Friedrich Wilhelms Universität Bonn, Germany, 2004.

[388] André Nicoll, Jan Meseth, Gero Müller, and Reinhard Klein. Fractional Fourier Texture Masks: Guiding Near-Regular Texture Synthesis. *Computer Graphics Forum*, 24(3):569–579, 2005.

[389] Kasper Høi Nielsen and Niels Jørgen Christensen. Real-Time Recursive Specular Reflections on Planar and Curved Surfaces using Graphics Hardware. *Journal of WSCG*, 10(3):91–98, 2002.

[390] Mangesh Nijasure, Sumanta N. Pattanaik, and Vineet Goel. Real-time Global Illumination on GPUs. *Journal of Graphics Tools*, 10(2):555–571, 2005.

[391] Ko Nishino, Yoichi Sato, and Katsushi Ikeuchi. Eigen-Texture Method: Appearance Compression and Synthesis Based on a 3D Model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1257–1265, 2001.

[392] NVidia. Mipmapping Normal Maps. available from NVidia website, 2004.

[393] NVidia. Shader Model 3.0 Unleashed. SIGGRAPH exhibitors session, 2004.

[394] Eyal Ofek and Ari Rappoport. Interactive reflections on curved objects. In *Proc. of SIGGRAPH 1998*, pages 333–342, 1998.

[395] Eisaku Ohbuchi. A Real-Time Refraction Renderer for Volume Objects Using a Polygon-Rendering Scheme. In *Computer Graphics International*, pages 190–195, 2003.

[396] Marc Olano and Michael North. Normal Distribution Mapping. Technical Report UNC CSTR 97-041, University of North Carolina, Chapel Hill, 1997.

[397] Gustavo Oliveira. Refractive Texture Mapping, Part Two. http://www.gamasutra.com/features/20001117/oliveira_01.htm, 2000.

[398] Manuel M. Oliveira. *Relief Texture Mapping*. PhD thesis, University of North Carolina, Chapel Hill, 2000.

[399] Michael Oren and Shree K. Nayar. Generalization of Lambert's Reflectance Model. In *Proc. of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 239–246, 1994.

[400] Carol O'Sullivan, Sarah Howlett, Rachel McDonnell, Yann Morvan, and Keith O'Conor. Perceptually Adaptive Graphics. In *Eurographics State of the Art Reports*, pages 141–162, 2004.

[401] Ryan Overbeck, Aner Ben-Artzi, Ravi Ramamoorthi, and Eitan Grinspun. Exploiting Temporal Coherence for Incremental All-Frequency Relighting. In *Rendering Techniques*, pages 151–160, 2006.

[402] Rupert Paget and I. D. Longstaff. Texture Synthesis via a Noncausal Nonparametric Multiscale Markov Random Field. *IEEE Transactions on Image Processing*, 7(6):925–931, 1998.

[403] James Painter and Kenneth Sloan. Antialiased Ray Tracing by Adaptive Progressive Refinement. *Computer Graphics*, 23(3):281–288, 1989.

[404] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive Ray Tracing for Isosurface Rendering. In *Proc. of Visualization '98*, pages 233–238, 1998.

[405] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive Ray Tracing. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 119–126, April 1999.

[406] Sumanta N. Pattanaik, James A. Ferwerda, Mark D. Fairchild, and Donald P. Greenberg. A Multiscale Model of Adaptation and Spatial Vision for Realistic Image Display. In *Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 287–298, 1998.

[407] Sumanta N. Pattanaik, James A. Ferwerda, Kenneth E. Torrance, and Donald P. Greenberg. Validation of Global Illumination Solutions through CCD Camera Measurements. In *Proc. of the Fifth Color Imaging Conference*, pages 250–253, 1997.

[408] Sumanta N. Pattanaik and Sudhir P. Mudur. Adjoint Equations and Random Walks for Illumination Computation. *ACM Transactions on Graphics*, 14(1):77–102, 1995.

[409] Fabio Pellacini, James A. Ferwerda, and Donald P. Greenberg. Toward a Psychophysically-Based Light Reflection Model for Image Synthesis. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 55–64, 2000.

[410] Ken Perlin. An Image Synthesizer. In *Proc. of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, pages 287–296, 1985.

[411] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The VolumePro Real-Time Ray-Casting System. In *Proc. of the 26th annual conference on Computer graphics and interactive techniques*, pages 251–260, 1999.

[412] Matt Pharr and Greg Humphreys. *Physically Based Rendering*. Elsevier Inc., 2004.

[413] Bui Tuong Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[414] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer, 2nd edition, 1997.

[415] Erik Pojar and Dieter Schmalstieg. User-Controlled Creation of Multiresolution Meshes. In *Proc. of the Symposium on Interactive 3D graphics*, pages 127–130, 2003.

[416] Fábio Policarpo and Manuel M. Oliveira. Relief Mapping of Non-Height-Field Surface Details. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 55–62, 2006.

[417] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 155–162, 2005.

[418] Voicu Popescu, ChunhuiMei, Jordan Dauble, and Elisha Sacks. Reflected-Scene Impostors for Realistic Reflections at Interactive Rates. *Computer Graphics Forum*, 25(3):313–322, 2006.

[419] Jovan Popović and Hugues Hoppe. Progressive Simplicial Complexes. In *Proc. of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, pages 217–224, 1997.

[420] Pierre Poulin and Alain Fournier. A Model for Anisotropic Reflection. *Computer Graphics*, 24(4):273–282, 1990.

[421] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped Textures. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 465–470. ACM SIGGRAPH, July 2000.

[422] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C Second Edition*. Cambridge University Press, New York, 1992.

[423] Jan Přikryl. *Radiosity Methods Driven by Human Perception*. PhD thesis, Technische Universität Wien, Austria, 2001.

[424] Chris Prince. Progressive Meshes for Large Models of Arbitrary Topology. Master's thesis, University of Washington, 2000.

[425] Enrico Puppo. Variable Resolution Triangulations. *Computational Geometry: Theory and Applications*, 11(3–4):219–238, 1998.

[426] Timothy Purcell. The SHARP Ray Tracing Architecture. SIGGRAPH Course on Interactive Ray Tracing, 2001.

[427] Timothy J. Purcell. *Ray Tracing on a Stream Processor*. PhD thesis, Stanford University, CA, March 2004.

[428] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics*, 21(3):703–712, 2002.

[429] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon Mapping on Programmable Graphics Hardware. In *Proc. of Graphics Hardware*, pages 41–50, 2003.

[430] Paul Rademacher, Jed Lengyel, Edward Cutrell, and Turner Whitted. Measuring the Perception of Visual Realism in Images. In *Proc. of the 12th Eurographics Workshop on Rendering Techniques*, pages 235–248, 2001.

[431] Ravi Ramamoorthi and Pat Hanrahan. An Efficient Representation for Irradiance Environment Maps. In *Proc. of SIGGRAPH 2001*, pages 497–500, 2001.

[432] Ravi Ramamoorthi and Pat Hanrahan. Frequency Space Environment Map Rendering. *ACM Transactions on Graphics*, 21(3):517–526, 2002.

[433] Ganesh Ramanarayanan, Kavita Bala, and Bruce Walter. Feature-Based Textures. In *Proc. of the 15th Eurographics Workshop on Rendering*, pages 265–274, 2004.

[434] Mahesh Ramasubramanian, Sumanta N. Pattanaik, and Donald P. Greenberg. A Perceptually Based Physical Error Metric for Realistic Image Synthesis. In *Proc. of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 73–82, 1999.

[435] RealReflect. Website. www.realreflect.org, 2005.

[436] Erik Reinhard, Greg Ward, Sumanta Pattanaik, and Paul Debevec. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*. Morgan Kaufman, 1st edition, 2005.

[437] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-Level Ray Tracing Algorithm. *ACM Transactions on Graphics*, 24(3):1176–1185, 2005.

[438] Eleanor G. Rieffel and Wolfgang Polak. An Introduction to Quantum Computing for Non-Physicists. *ACM Computing Surveys*, 32(3):300–335, 2000.

[439] Alyn Rockwood, Kurt Heaton, and Tom Davis. Real-Time Rendering of Trimmed Surfaces. *Computer Graphics*, 23(3):107–116, 1989.

[440] David Roger and Nicolas Holzschuch. Accurate Specular Reflections in Real-Time. *Computer Graphics Forum*, 25(3):293–302, 2006.

[441] Jarek R. Rossignac and Paul Borrel. Multi-Resolution 3D Approximations for Rendering Complex Scenes. In *Geometric Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, Berlin, 1993.

[442] Holly Rushmeier, Gabriel Taubin, and André Guéziec. Applying Shape from Lighting Variation to Bump Map Capture. In *Rendering Techniques*, pages 35–44, 1997.

[443] Holly Rushmeier, Greg Ward, Christine Piatko, Phil Sanders, and Bert Rust. Comparing Real and Synthetic Images: Some Ideas About Metrics. In *Proc. of Rendering Workshop*, pages 82–91, Dublin, Ireland, 1995.

[444] Szymon Rusinkiewicz. A New Change of Variables for Efficient BRDF Representation. In *Rendering Techniques*. Springer-Verlag, Wien, Austria, 1998.

[445] Szymon Rusinkiewicz and Marc Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 343–352, 2000.

[446] Pedro V. Sander, Z. J. Wood, Steven J. Gortler, John Snyder, and Hugues Hoppe. Multi-Chart Geometry Images. In *Proc. of Symposium on Geometry Processing*, pages 146–155, 2003.

[447] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Efficient and Realistic Visualization of Cloth. In *Rendering Techniques*, Leuven, Belgium, June 2003.

[448] Gernot Schaufler. Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes. In *Rendering Techniques*, pages 151–162, 1997.

[449] Gernot Schaufler. Per-Object Image Warping with Layered Impostors. In *Rendering Techniques*, pages 145–156, 1998.

[450] Sagi Schein, Eran Karpen, and Gershon Elber. Real-time geometric deformation displacement maps using programmable hardware. *The Visual Computer*, 21(8):791–800, 2005.

[451] Andreas Schilling. Towards Real-Time Photorealistic Rendering: Challenges and Solutions. In *Proc. of the Workshop on Graphics Hardware*, pages 7–15, 1997.

[452] Andreas Schilling and Reinhard Klein. Rendering of multiresolution models with texture. *Computers & Graphics*, 22(6):667–674, 1998.

[453] Christophe Schlick. An Inexpensive BRDF Model for Physically-based Rendering. *Computer Graphics Forum*, 13(3):233–246, 1994.

[454] Jörg Schmittler, Alexander Leidinger, and Philipp Slusallek. A Virtual Memory Architecture for Real-Time Ray Tracing Hardware. *Computer and Graphics*, 27(5):693–699, 2003.

[455] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. SaarCOR: A Hardware Architecture for Ray Tracing. In *Proc. of Graphics Hardware*, pages 27–36, 2002.

[456] Jörg Schmittler, Sven Woop, Daniel Wagner, Wolfgang J. Paul, and Philipp Slusallek. Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip. In *Proc. of Graphics Hardware*, pages 95–106, 2004.

[457] Martin Schneider. Effiziente Verfahren zur Photorealistischen Materialdarstellung unter Verwendung der Bidirektionalen Texturfunktion. Master's thesis, Rheinische Friedrich Wilhelms Universität Bonn, Germany, 2004.

[458] Martin Schneider. Real-Time BTF Rendering. In *Proc. of CESCG 2004*, April 2004.

[459] Roland Schregle and Jan Wienold. Physical Validation of Global Illumination Methods: Measurement and Error Analysis. *Computer Graphics Forum*, 23(4):761–781, 2004.

[460] Peter Schröder and Wim Sweldens. Spherical Wavelets: Efficiently Representing Functions on the Sphere. In *Proc. of SIGGRAPH*, pages 161–172, 1995.

[461] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. *Computer Graphic*, 26(2):65–70, 1992.

[462] Roberto Scopigno, Carlos Andujar, Michael Goesele, and Hendrik Lensch. 3D Data Acquisition. Eurographics Tutorial 1, 2002.

[463] Helge Seetzen, Wolfgang Heidrich, Wolfgang Stuerzlinger, Greg Ward, Lorne Whitehead, Matthew Trentacoste, Abhijeet Ghosh, and Andrejs Vorozcovs. High Dynamic Range Display Systems. *ACM Transactions on Graphics*, 23(3):760–768, 2004.

[464] Helge Seetzen, Lorne Whitehead, and Greg Ward. A High Dynamic Range Display Using Low and High Resolution Modulators. *SID Symposium Digest of Technical Papers*, 34(1):1450–1453, 2003.

[465] Benjamin Segovia, Jean-Claude Iehl, Richard Mitanchey, and Bernard Péroche. Bidirectional Instant Radiosity. In *Rendering Techniques*, pages 389–397, 2006.

[466] Jonathan Shade, Steven Gortler, Li-Wei He, and Richard Szeliski. Layered Depth Images. In *Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 231–242, 1998.

[467] Michael Shafae and Renato Pajarola. DStrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering. In *Proc. of Pacific Graphics*, pages 271–280, 2003.

[468] Eric Shaffer and Michael Garland. A Multiresolution Representation for Massive Meshes. *Transactions on Visualization and Computer Graphics*, 11(2):139–148, 2005.

[469] Musawir A. Shah and Sumanta Pattanaik. Caustics Mapping: An Image-space Technique for Real-time Caustics. Technical Report CS TR 50-07, School of Engineering and Computer Science, University of Central Florida, 2005.

[470] Ariel Shamir, Chandrajit Bajaj, and Valerio Pascucci. Multi-Resolution Dynamic Meshes with Arbitrary Deformations. In *Proc. of Visualization*, pages 423–430, 2000.

[471] Ariel Shamir and Valerio Pascucci. Temporal and spatial level of details for dynamic meshes. In *Proc. of Symposium on Virtual Reality Software and Technology*, pages 77–84, 2001.

[472] Alla Sheffer. Model Simplification for Meshing using Face Clustering. *Computer-Aided Design*, 33:925–934, 2001.

[473] Hamid R. Sheikh, Alan C. Bovik, and Gustavo de Veciana. An Information Fidelity Criterion for Image Quality Assessment Using Natural Scene Statistics. *IEEE Transactions on Image Processing*, 14(12):2117–2129, 2005.

[474] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and Approximating Implicit Surfaces from Polygon Soup. *ACM Transactions on Graphics*, 23(3):896–904, 2004.

[475] Peter Shirley. A Ray Tracing Method for Illumination Calculation in Diffuse-Specular Scenes. In *Proc. of Graphics Interface*, pages 205–212, May 1990.

[476] Peter Shirley. Radiosity via Ray Tracing. In *Graphics Gems II*. Academic Press Professional, Boston, MA, 1991.

[477] Peter Shirley and R. Keith Morley. *Realistic Ray Tracing*. A K Peters, Ltd., 2nd edition, 2003.

[478] Peter Shirley, Brian Smits, Helen Hu, and Eric Lafortune. A Practitioners' Assessment of Light ReflectionModels. In *Proc. of the 5th Pacific Conference on Computer Graphics and Applications*, pages 40–49, 1997.

[479] Leon A. Shirman and Salim S. Abi-Ezzi. The Cone of Normals Technique for Fast Processing of Curved Patches. *Computer Graphics Forum*, 12(3):261–272, 1993.

[480] Le-Jeng Shiue, Ian Jones, and Jörg Peters. A Realtime GPU Subdivision Kernel. *ACM Transactions on Graphics*, 24(3):1010–1015, 2005.

[481] Heung Yeung Shum, Yin Li, and Sing Bing Kang. An Introduction to Image-Based Rendering. In *Integrated image and graphics technologies*, pages 131–159. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[482] François X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A General Two-Pass Method Integrating Specular and Diffuse Reflection. *Computer Graphics*, 23(3):335–344, 1989.

[483] François X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A Global Illumination Solution for General Reflectance Distributions. *Computer Graphics*, 25(4):187–196, 1991.

[484] Maryann Simmons and Carlo H. Séquin. Tapestry: A Dynamic Mesh-based Display Representation for Interactive Rendering. In *Proc. of the Eurographics Workshop on Rendering Techniques*, pages 329–340, 2000.

[485] Simoncelli and Portilla. Texture Characterization via Joint Statistics of Wavelet Coefficient Magnitudes. In *Proc. of International Conference on Image Processing – Volume 1*, pages 62–66, 1998.

[486] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. Clustered Principal Components for Precomputed Radiance Tansfer. *ACM Transactions on Graphics*, 22(3):382–391, 2003.

[487] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Transactions on Graphics*, 21(3):527–536, 2002.

[488] Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. Bi-Scale Radiance Transfer. *ACM Transactions on Graphics*, 22(3):370–375, 2003.

[489] Peter-Pike Sloan, Ben Luna, and John Snyder. Local, Deformable Precomputed Radiance Transfer. *ACM Transactions on Graphics*, 24(3):1216–1224, 2005.

[490] Brian Smits and Henrik W. Jensen. Global Illumination Test Scenes. Technical Report UUCS-00-013, University of Utah, 2000.

[491] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical Pattern Mapping. *ACM Transactions on Graphics*, 21(3):673–680, 2002.

[492] Petr Somol and Michal Haindl. Novel Path Search Algorithm for Image Stitching and Advanced Texture Tiling. In *Proc. of WSCG*, pages 155–, 2005.

[493] Ying Song, Yanyun Chen, Xin Tong, Stephen Lin, Jiaoying Shi, Baining Guo, and Heung-Yeung Shum. Shell Radiance Texture Functions. *The Visual Computer*, 21(8–10):774–782, 2005.

[494] Taigo Sousa. Generic Refraction Simulation. In *GPU Gems 2*, pages 295–305. NVidia, 2005.

[495] R. Sriram, Joseph M. Francos, and William A. Pearlman. Texture Coding Using a Wold Decomposition Model. *IEEE Transactions on Image Processing*, 5(9):1382–1386, 1996.

[496] Jos Stam. Random Caustics: Natural Textures and Wave Theory Revisited. In *ACM SIGGRAPH Visual Proceedings: The art and interdisciplinary programs of SIGGRAPH '96*, page 150, 1996.

[497] Jos Stam. Diffraction Shaders. In *Proc. of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 101–110, 1999.

[498] Marc Stamminger, Jörg Haber, Hartmut Schirmacher, and Hans-Peter Seidel. Walkthroughs with Corrective Texturing. In *Proc. of the Eurographics Workshop on Rendering Techniques*, pages 377–388, 2000.

[499] William A. Stokes, James A. Ferwerda, Bruce Walter, and Donald P. Greenberg. Perceptual Illumination Components: A New Approach to Efficient, High Quality Global Illumination Rendering. *ACM Transaction on Graphics*, 23(3):742–749, 2004.

[500] Gilbert Strang and Truong Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[501] Jessi Stumpfel, Chris Tchou, Andrew Jones, Tim Hawkins, Andreas Wenger, and Paul Debevec. Direct HDR Capture of the Sun and Sky. In *Proc. of the 3rd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pages 145–149, 2004.

[502] Wolfgang Stürzlinger and Rui Bastos. Interactive Rendering of Globally Illuminated Glossy Scenes. In *Proc. of the Eurographics Workshop on Rendering Techniques*, pages 93–102, 1997.

[503] Kartic S. Subr, Meenakshisundaram Gopi, Renato Pajarola, and Miguel Sainz. Point Light Field for Point Rendering Systems. Technical Report UCI-ICS-03-28, Department of Computer Science, University of California Irvine, 2003.

[504] Yinlong Sun. Self Shadowing and Local Illumination of Randomly Rough Surfaces. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition – volume 1*, pages 158–165, 2004.

[505] Ben Sunshine-Hill and Petros Faloutsos. Photorealistic Lighting with Offset Radiance Transfer Mapping. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 15–21, 2006.

[506] Herb Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3), 2005.

[507] Frank Suykens, Karl vom Berge, Ares Lagae, and Philip Dutré. Interactive Rendering of Bidirectional Texture Functions. In *Eurographics 2003*, pages 463–472, September 2003.

[508] László Szirmay-Kalos, Barnabás Aszódi, István Lazányi, and Mátyás Premecz. Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum*, 24(3):171–176, 2005.

[509] László Szirmay-Kalos, László Kovás, and Ali M. Abbas. Testing Monte-Carlo Global Illumination Methods with Analytically Computable Scenes. In *Proc. of Winter School of Computer Graphics*, pages 419–426, 2001.

[510] László Szirmay-Kalos and Werner Purgathofer. Global Ray-Bundle Tracing with Hardware Acceleration. In *Proc. of the 9th Eurographics Workshop on Rendering*, pages 247–258, 1998.

[511] László Szirmay-Kalos and Werner Purgathofer. Analysis of the Quasi-Monte Carlo Integration of the Rendering Equation. In *Proc. of WSCG*, pages 281–288, 1999.

[512] Atsushi Takagi, Hitoshi Takaoka, Tetsuya Oshima, and Yoshinori Ogata. Accurate Rendering Technique Based on Colorimetric Conception. *Computer Graphics*, 24(4):263–272, 1990.

[513] Ping Tan, Stephen Lin, Long Quan, Baining Guo, and Heung-Yeung Shum. Multiresolution Reflectance Filtering. In *Rendering Techniques*, pages 111–116, 2005.

[514] Marco Tarini and Paolo Cignoni. Pinchmaps: Textures with Customizable Discontinuities. *Computer Graphics Forum*, 24(3):557–568, 2005.

[515] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. PolyCube-Maps. *ACM Transactions on Graphics*, 23(3):853–860, 2004.

[516] Seth Teller, Kavita Bala, and Julie Dorsey. Conservative Radiance Interpolants for Ray Tracing. In *Proc. of the Eurographics Workshop on Rendering Techniques*, pages 257–268, 1996.

[517] Niels Thrane and Lars Ole Simonsen. A Comparison of Acceleration Structures for GPU Assisted Ray Tracing. Master's thesis, University of Aarhus, Denmark, 2005.

[518] Parag Tole, Fabio Pellacini, Bruce Walter, and Donald P. Greenberg. Interactive global illumination in dynamic scenes. *ACM Transactions on Graphics*, 21(3):537–546, July 2002.

[519] Xin Tong, Jiaping Wang, Stephen Lin, Baining Guo, and Heung-Yeung Shum. Modeling and Rendering of Quasi-Homogeneous Materials. *ACM Transactions on Graphics*, 24(3):1054–1061, 2005.

[520] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces. *ACM Transactions on Graphics*, 21(3):665–672, July 2002.

[521] Kenneth E. Torrance and E. M. Sparrow. Theory for Off-Specular Reflection from Rough Surfaces. *Journal of the Optical Society of America*, 57(9):1105–1114, 1967.

[522] Chris Trendall and A. James Stewart. General Calculations using Graphics Hardware with Applications to Interactive Caustics. In *Proc. of the Eurographics Workshop on Rendering Techniques*, pages 287–298, 2000.

[523] Yu-Ting Tsai and Zen-Chung Shih. All-Frequency Precomputed Radiance Transfer using Spherical Radial Basis Functions and Clustered Tensor Approximation. *ACM Transactions on Graphics*, 25(3):967–976, 2006.

[524] Greg Turk. Generating Textures on Arbitrary Surfaces using Reaction-Diffusion. *Computer Graphics*, 25(4):289–298, 1991.

[525] Greg Turk. Re-Tiling Polygonal Surfaces. *Computer Graphics*, 26(2):55–64, 1992.

[526] Greg Turk. Texture Synthesis on Surfaces. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 347–354, 2001.

[527] Tushar Udeshi and Charles D. Hansen. Towards Interactive Photorealistic Rendering of Indoor Scenes: A Hybrid Approach. In *Rendering Techniques*, pages 63–76, 1999.

[528] Christiane Ulbricht, Alexander Wilkie, and Werner Purgathofer. Verification of Physically Based Rendering Algorithms. In *Eurographics State of the Art Reports*, pages 95–112, 2005.

[529] Thomas Ullmann, Daniel Beier, Beat Bruderlin, and Alexander Schmidt. Adaptive Progressive Vertex Tracing in Distributed Environments. In *Proc. of Ninth Pacific Conference on Computer Graphics and Applications*, pages 285–294, 2001.

[530] Thomas Ullmann and Beat Bruderlin. Adaptive Progressive Vertex-Tracing for Interactive Reflections. In *Eurographics Short Papers*, 2001.

[531] Gokul Varadhan and Dinesh Manocha. Out-of-core rendering of massive geometric models. In R. Moorhead, Markus Gross, and Kenneth I. Joy, editors, *Proc. of IEEE Visualization 2002*, pages 69–76, Boston, Massachussetts, 2002.

[532] M. Alex O. Vasilescu and Demetri Terzopoulos. TensorTextures: Multilinear Image-Based Rendering. *ACM Transactions on Graphics*, 23(3):336–342, 2004.

[533] Eric Veach and Leonidas J. Guibas. Bidirectional Estimators for Light Transport. In *Proc. of Eurographics Rendering Workshop*, pages 147–162, June 1994.

[534] Emmanuel Viale and Kelly Dempski. *Advanced Lighting and Materials With Shaders*. Wordware Publishing, 2004.

[535] Daniel Vlasic, Hanspeter Pfister, Sergey Molinov, Radek Grzeszczuk, and Wojciech Matusik. Opacity Light Fields: Interactive Rendering of Surface Light Fields with View-Dependent Opacity. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 65–74, 2003.

[536] Carsten Waechter and Alexander Keller. Instant Ray Tracing: The Bounding Interval Hierarchy. In *Rendering Techniques*, pages 139–149, 2006.

[537] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, Saarbrücken, Germany, 2004.

[538] Ingo Wald, Carsten Benthin, Alexander Efremov, Tim Dahmen, Johannes Günther, Andreas Dietrich, Vlastimil Havran, Philipp Slusallek, and Hans-Peter Seidel. A Ray Tracing based Virtual Reality Framework for Industrial Design. Technical Report UUSCI-2005-009, University of Utah, SCI Institute, 2005.

[539] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 20(3):153–164, 2001.

[540] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proc. of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, page 11, 2003.

[541] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. Technical Report, SCI Institute, University of Utah, No UUSCI-2005-014, 2006.

[542] Ingo Wald, Andreas Dietrich, and Philipp Slusallek. An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models. In *Rendering Techniques*, pages 81–92, 2004.

[543] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics*, 25(3):485–493, 2006.

[544] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive Global Illumination using Fast Ray Tracing. In *Proc. of the 13th Eurographics Workshop on Rendering*, pages 15–24, 2002.

[545] Ingo Wald, Timothy J. Purcell, Jörg Schmittler, Carsten Benthin, and Philipp Slusallek. Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports*, 2003.

[546] Ingo Wald and Philipp Slusallek. State-of-the-Art in Interactive Ray-Tracing. In *Eurographics State of the Art Reports*, pages 21–42, 2001.

[547] Ingo Wald, Philipp Slusallek, and Carsten Benthin. Interactive Distributed Ray Tracing of Highly Complex Models. In *Rendering Techniques*, pages 274–285, 2001.

[548] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods. *Computer Graphics*, 21(4):311–320, 1987.

[549] Bruce Walter, George Drettakis, and Donald P. Greenberg. Enhancing and Optimizing the Render Cache. In *Rendering Techniques*, pages 37–42, 2002.

[550] Bruce Walter, George Drettakis, and Steven Parker. Interactive Rendering using the Render Cache. In *Rendering Techniques*, volume 10, pages 235–246, June 1999.

[551] Michael Wand and Wolfgang Straßer. Multi-Resolution Point-Sample Raytracing. In *Graphics Interface*, pages 139–148, 2003.

[552] Michael Wand and Wolfgang Straßer. Real-Time Caustics. *Computer Graphics Forum*, 22(3):611–620, 2003.

[553] Hongcheng Wang, Qing Wu, Lin Shi, Yizhou Yu, and Narendra Ahuja. Out-of-Core Tensor Approximation of High Dimensional Visual Data. *ACM Transactions on Graphics*, 24(3):527–535, 2005.

[554] Jiaping Wang, Xin Tong, John Snyder, Yanyun Chen, Baining Guo, and Heung-Yeung Shum. Capturing and rendering geometry details for BTF-mapped surfaces. *The Visual Computer*, 21(8–10):559–568, 2005.

[555] Jiaping Wang, Kun Xu, Kun Zhou, Stephen Lin, Shimin Hu, and Baining Guo. Spherical Harmonics Scaling. *The Visual Computer*, 22(9):713–720, 2006.

[556] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. View-Dependent Displacement Mapping. *ACM Transactions on Graphics*, 22(3):334–339, 2003.

[557] Lujin Wang, Xianfeng Gu, Klaus Mueller, and Shing-Tung Yau. Uniform texture synthesis and texture mapping using global parameterization. *The Visual Computer*, 21(8–10):801–810, 2005.

[558] Rui Wang, Ren Ng, David Luebke, and Greg Humphreys. Efficient Wavelet Rotation for Environment Map Rendering. In *Rendering Techniques*, pages 173–182, 2006.

[559] Rui Wang, John Tran, and David Luebke. All-Frequency Interactive Relighting of Translucent Objects with Single and Multiple Scattering. *ACM Transactions on Graphics*, 24(3):1202–1207, 2005.

[560] Rui Wang, John Tran, and David P. Luebke. All-Frequency Relighting of Non-Diffuse Objects using Separable BRDF Approximation. In *Rendering Techniques*, pages 345–354, 2004.

[561] Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. Generalized Displacement Maps. In *Rendering Techniques*, pages 227–233, 2004.

[562] Zhou Wang, Alan C. Bovik, and Ligang Lu. Why is image quality assessment so difficult? In *Proc. of the 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3313–3316, 2002.

[563] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eorn P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[564] Gregory Ward and Maryann Simmons. The Holodeck Ray Cache: An Interactive Rendering System for Global Illumination in Nondiffuse Environments. *ACM Transactions on Graphics*, 18(4):361–368, 1999.

[565] Gregory J. Ward. Measuring and Modeling Anisotropic Reflection. *Computer Graphics*, 26(2):265–272, 1992.

[566] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A Ray Tracing Solution for Diffuse Interreflection. *Computer Graphics*, 22(4):85–92, 1988.

[567] Gregory Ward Larson. The Holodeck: A Parallel Ray-caching Rendering System. In *Proc. of the 2nd Eurographics Workshop on Parallel Graphics and Visualisation*, September 1998.

[568] Andrew B. Watson, James Hu, and John F. McGowan. DVQ: A digital video quality metric based on human vision. *Electronic Imaging*, 10(1):20–29, 2001.

[569] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques*. ACM Press New York / Addison-Wesley, 1992.

[570] Web3D Consortium. Extensible 3D (X3D). ISO/IEC 19775:2004, 2004.

[571] Li-Yi Wei. Tile-Based Texture Mapping on Graphics Hardware. In *Proc. of Graphics Hardware*, pages 55–63, 2004.

[572] Li-Yi Wei and Marc Levoy. Fast Texture Synthesis using Tree-Structured Vector Quantization. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 479–488, 2000.

[573] Li-Yi Wei and Marc Levoy. Texture Synthesis Over Arbitrary Manifold Surfaces. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 355–360, 2001.

[574] Li-Yi Wei and Marc Levoy. Order-Independent Texture Synthesis. Technical Report TR-2002-01, Stanford University, Computer Science Department, 2002.

[575] Daniel Weiskopf, Tobias Schafhitzel, and Thomas Ertel. GPU-Based Nonlinear Ray Tracing. *Computer Graphics Forum*, 23(3):625–634, 2004.

[576] Eric W. Weisstein et al. Fractional Fourier Transform. From MathWorld–A Wolfram Web Resource. mathworld.wolfram.com/FractionalFourierTransform.html, 1999.

[577] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting Reflectance Functions from Complex Surfaces. *Computer Graphics*, 26(2):255–264, 1992.

[578] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[579] Alexander Wilkie, Robert F. Tobler, Christiane Ulbricht, Georg Zotti, and Werner Purgathofer. An Analytical Model for Skylight Polarisation. In *Proc. of the Eurographics Symposium on Rendering*, pages 387–399, 2004.

[580] Nathaniel Williams, David Luebke, Jonathan D. Cohen, Michael Kelley, and Brenden Schubert. Perceptually Guided Simplification of Lit, Textured Meshes. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 113–121, 2003.

[581] Andrew Wilson and Dinesh Manocha. Simplifying Complex Environments using Incremental Textured Depth Meshes. *ACM Transactions on Graphics*, 22(3):678–688, 2003.

[582] Stefan Winkler. Visual Fidelity and Perceived Quality: Towards Comprehensive Metrics. In *Proc. of SPIE Human Vision and Electronic Imaging*, pages 114–125, 2001.

[583] Andrew Witkin and Michael Kass. Reaction-Diffusion Textures. *Computer Graphics*, 25(4):299–308, 1991.

[584] Tien-Tsin Wong, Pheng-Ann Heng, Siu-Hang Or, and Wai-Yin Ng. Image-based Rendering with Controllable Illumination. In *Rendering Techniques*, pages 13–22, 1997.

[585] Andrew Wood, B. McCane, and S. A. King. Ray Tracing Arbitrary Objects on the GPU. In *Proc. of Image and Vision Computing New Zealand*, pages 327–332, November 2004.

[586] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface Light Fields for 3D Photography. In *Proc. of SIGGRAPH*, pages 287–296, 2000.

[587] Sven Woop, Jörg Schmittler, and Philipp Slusallek. RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing. *ACM Transactions on Graphics*, 24(3):434–444, 2005.

[588] Hongzhi Wu, Li-Yi Wei, Xi Wang, and Baining Guo. Silhouette Texture. In *Rendering Techniques*, pages 285–296, 2006.

[589] Jianhua Wu and Leif Kobbelt. Fast Mesh Decimation by Multiple-Choice Techniques. In *Proc. of Vision, Modeling, and Visualization*, pages 241–248, 2002.

[590] Jianhua Wu and Leif Kobbelt. A Stream Algorithm for the Decimation of Massive Meshes. In *Proc. of Graphics Interface*, pages 185–192, 2003.

[591] Jianhua Wu and Leif Kobbelt. Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.

[592] Qing Wu and Yizhou Yu. Feature Matching and Deformation for Texture Synthesis. *ACM Transactions on Graphics*, 23(3):364–367, 2004.

[593] Chris Wyman. An Approximate Image-Space Approach for Interactive Refraction. *ACM Transactions on Graphics*, 24(3):1050–1053, 2005.

[594] Chris Wyman. Interactive Image-Space Refraction of Nearby Geometry. In *Proc. of GRAPHITE*, pages 205–211, 2005.

[595] Chris Wyman and Scott Davis. Interactive Image-Space Techniques for Approximating Caustics. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 153–160, 2006.

[596] Chris Wyman, Charles Hansen, and Peter Shirley. Interactive Caustics Using Local Precomputed Irradiance. In *Proc. of the Pacific Conference on Computer Graphics and Applications*, pages 143–151, 2004.

[597] Julie C. Xia, Jihad El-Sana, and Amitabh Varshney. Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, 1997.

[598] Julie C. Xia and Amitabh Varshney. Dynamic View-Dependent Simplification for Polygonal Models. In *Proc. of IEEE Visualization*, pages 335–344, 1996.

[599] Xinyu Xiang, Martin Held, and Joseph S. B. Mitchell. Fast and Effective Stripification of Polygonal Surface Models. In *Proc. of the Symposium on Interactive 3D Graphics*, pages 71–78, 1999.

[600] Ying-Qing Xu, Baining Guo, and Harry Shum. Chaos Mosaic: Fast and Memory Efficient Texture Synthesis. Technical Report MSR-TR-2000-32, Microsoft Research, 2000.

[601] Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and Shape Synthesis on Surfaces. In *Rendering Techniques*, pages 301–312, June 2001.

[602] Sung-Eui Yoon, Christian Lauterbach, and Dinesh Manocha. R-LODs: Fast LOD-based Ray Tracing of Massive Models. *The Visual Computer*, 22(9):772–784, 2006.

[603] Sung-Eui Yoon, Peter Lindstrom, Valerio Pascucci, and Dinesh Manocha. Cache-Oblivious Mesh Layouts. *ACM Transactions on Graphics*, 24(3):886–893, 2005.

[604] Sung-Eui Yoon, Brian Salomon, Russell Gayle, and Dinesh Manocha. Quick-VDR: Out-of-Core View-Dependent Rendering of Gigantic Models. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):369–382, 2005.

[605] Akiko Yoshida, Volker Blanz, Karol Myszkowski, and Hans-Peter Seidel. Perceptual Evaluation of Tone Mapping Operators with Real-World Scenes. In *Human Vision and Electronic Imaging X, IS&T/SPIE's 17th Annual Symposium on Electronic Imaging*, pages 192–203, 2005.

[606] Jingyi Yu, Jason Yang, and Leonard McMillan. Real-time Reflection Mapping with Parallax. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, pages 133–138, 2005.

[607] Steve Zelinka, Hui Fang, Michael Garland, and John C. Hart. Interactive Material Replacement in Photographs. In *Proc. of the Conference on Graphics Interface*, pages 227–232, 2005.

[608] Steve Zelinka and Michael Garland. Permission Grids: Practical, Error-Bounded Simplification. *ACM Transactions on Graphics*, 21(2):207–229, 2002.

[609] Steve Zelinka and Michael Garland. Towards Real-Time Texture Synthesis with the Jump Map. In *Rendering Techniques*, Pisa, Italy, June 2002.

[610] Steve Zelinka and Michael Garland. Interactive Texture Synthesis on Surfaces using Jump Maps. In *Rendering Techniques*, pages 90–96, 2003.

[611] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics*, 22(3):295–302, 2003.