Realistic Visualization of Animated Virtual Cloth

Dissertation

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

vorgelegt von

Dipl.-Geophys. Mirko Sattler
aus Ratzeburg

Bonn, Dezember 2006

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

Dekan:	Prof. Dr. Armin B. Cremers, Universität Bonn
1. Referent:	Prof. Dr. Reinhard Klein, Universität Bonn
2. Referent:	Prof. Dr.-Ing. Philipp Slusallek, Universität Saarbrücken
Tag der Promotion:	26.10.2007

Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn
http://hss.ulb.uni-bonn.de/diss_online
elektronisch publiziert.

Erscheinungsjahr: 2007

This work is dedicated to my parents
Ingrid and Bernd-Uwe
for all their love & support.

If it looks right, it is right.

Acknowledgments

The presented work has been produced within the scope of the Computer Graphics group at the Institute of Computer Science II of the University of Bonn.

I wish to thank all the people who were directly or indirectly involved in the creation of this work. First of all, I wish to thank Prof. Reinhard Klein without his ideas and the fruitful discussions this work had not been possible. I also thank him for providing a pleasant working atmosphere and for giving an “outsider“ the chance to get into the field of computer graphics.

Furthermore, I thank all current and former members of the group, especially (in alphabetical order): Björn Ganster, Michael Guthe, Ferenc Kahlesz, Jan Meseth, Gero Müller, Marcin Novotni and Gabriel Zachmann and some colleagues from other universities, namely Jan Kautz, Markus Wacker, Stefan Kimmerle and Michael Keckeisen for joint publications and useful discussions.

Our secretary Simone Schäfer to keep up the good spirit in the group and introduction to the *Kölsche Lebensart*. My office fellows Ralf Sarlette and Gerhard Bendels, for the endless hours of fun we spent together. Ralf did a tremendous job creating and maintaining our laboratory.

In addition, I want to thank the Stanford 3D Scanning Repository for their great 3D models and all the people involved in the *Virtual-Try On* and the *RealReflect* project.

Thanks ;-)
Mirko, Bonn 2006

Abstract

Photo-realistic rendering of real-world objects is a broad research area with applications in various different areas, such as computer generated films, entertainment, e-commerce and so on. Within photo-realistic rendering, the rendering of cloth is a subarea which involves many important aspects, ranging from material surface reflection properties and macroscopic self-shadowing to animation sequence generation and compression.

In this thesis, besides an introduction to the topic plus a broad overview of related work, different methods to handle major aspects of cloth rendering are described.

Material surface reflection properties play an important part to reproduce the *look & feel* of materials, that is, to identify a material only by looking at it. The BTF (bidirectional texture function), as a function of viewing and illumination direction, is an appropriate representation of reflection properties. It captures effects caused by the mesostructure of a surface, like roughness, self-shadowing, occlusion, inter-reflections, subsurface scattering and color bleeding. Unfortunately a BTF data set of a material consists of hundreds to thousands of images, which exceeds current memory size of personal computers by far.

This work describes the first usable method to efficiently compress and decompress a BTF data for rendering at interactive to real-time frame rates. It is based on PCA (principal component analysis) of the BTF data set. While preserving the important visual aspects of the BTF, the achieved compression rates allow the storage of several different data sets in main memory of consumer hardware, while maintaining a high rendering quality.

Correct handling of complex illumination conditions plays another key role for the realistic appearance of cloth. Therefore, an upgrade of the BTF compression and rendering algorithm is described, which allows the support of distant direct HDR (high-dynamic-range) illumination stored in environment maps.

To further enhance the appearance, macroscopic self-shadowing has to be taken into account. For the visualization of folds and the life-like 3D impression, these kind of shadows are absolutely necessary. This work describes two methods to compute these shadows. The first is seamlessly integrated into the illumination part of the rendering algorithm and optimized for static meshes. Furthermore, another method is proposed, which allows the handling of dynamic objects. It uses hardware-accelerated occlusion queries for the visibility determination. In contrast to other algorithms, the presented algorithm, despite its simplicity, is fast

and produces less artifacts than other methods. As a plus, it incorporates changeable distant direct high-dynamic-range illumination.

The human perception system is the main target of any computer graphics application and can also be treated as part of the rendering pipeline. Therefore, optimization of the rendering itself can be achieved by analyzing human perception of certain visual aspects in the image. As a part of this thesis, an experiment is introduced that evaluates human shadow perception to speedup shadow rendering and provides optimization approaches.

Another subarea of cloth visualization in computer graphics is the animation of the cloth and avatars for presentations. This work also describes two new methods for automatic generation and compression of animation sequences.

The first method to generate completely new, customizable animation sequences, is based on the concept of finding similarities in animation frames of a given basis sequence. Identifying these similarities allows *jumps* within the basis sequence to generate endless new sequences.

Transmission of any animated 3D data over bandwidth-limited channels, like extended networks or to less powerful clients requires efficient compression schemes. The second method included in this thesis in the animation field is a geometry data compression scheme. Similar to the BTF compression, it uses PCA in combination with clustering algorithms to segment similar moving parts of the animated objects to achieve high compression rates in combination with a very exact reconstruction quality.

Zusammenfassung

Das photorealistisches Rendering realer Gegenstände ist ein weites Forschungsfeld und hat Anwendungen in vielen Bereichen. Dazu zählen Computer generierte Filme (CGI), die Unterhaltungsindustrie und E-Commerce. Innerhalb dieses Forschungsbereiches ist das Rendern von photorealistischer Kleidung ein wichtiger Bestandteil. Hier reichen die wichtigen Aspekte, die es zu berücksichtigen gilt, von optischen Materialeigenschaften über makroskopische Selbstabschattung bis zur Animationsgenerierung und -kompression.

In dieser Arbeit wird, neben der Einführung in das Thema, ein weiter Überblick über ähnlich gelagerte Arbeiten gegeben. Der Schwerpunkt der Arbeit liegt auf den wichtigen Aspekten der virtuellen Kleidungsvisualisierung, die oben beschrieben wurden.

Die optischen Reflektionseigenschaften von Materialoberflächen spielen eine wichtige Rolle, um das so genannte *look & feel* von Materialien zu charakterisieren. Hierbei kann ein Material vom Nutzer identifiziert werden, ohne dass er es direkt anfassen muss. Die BTF (bidirektionale Texturfunktion) ist eine Funktion die abhängig von der Blick- und Beleuchtungsrichtung ist. Daher ist sie eine angemessene Repräsentation von Reflektionseigenschaften. Sie enthält Effekte wie Rauheit, Selbstabschattungen, Verdeckungen, Interreflektionen, Streuung und Farbbluten, die durch die Mesostruktur der Oberfläche hervorgerufen werden. Leider besteht ein BTF Datensatz eines Materials aus hunderten oder tausenden von Bildern und sprengt damit herkömmliche Hauptspeicher in Computern bei weitem.

Diese Arbeit beschreibt die erste praktikable Methode, um BTF Daten effizient zu komprimieren, zu speichern und für Echtzeitanwendungen zum Visualisieren wieder zu dekomprimieren. Die Methode basiert auf der *Principal Component Analysis* (PCA), die Daten nach Signifikanz ordnet. Während die PCA die entscheidenen visuellen Aspekte der BTF erhält, können mit ihrer Hilfe Kompressionsraten erzielt werden, die es erlauben mehrere BTF Materialien im Hauptspeicher eines Consumer PC zu verwalten. Dies erlaubt ein High-Quality Rendering.

Korrektes Verwenden von komplexen Beleuchtungssituationen spielt eine weitere, wichtige Rolle, um Kleidung realistisch erscheinen zu lassen. Daher wird zudem eine Erweiterung des BTF Kompressions- und Renderingalgorithmuses erläutert, die den Einsatz von *High-Dynamic Range* (HDR) Beleuchtung erlaubt, die in *environment maps* gespeichert wird.

Um die realistische Erscheinung der Kleidung weiter zu unterstützen, muss die

makroskopische Selbstabschattung integriert werden. Für die Visualisierung von Falten und den lebensechten 3D Eindruck ist diese Art von Schatten absolut notwendig. Diese Arbeit beschreibt daher auch zwei Methoden, diese Schatten schnell und effizient zu berechnen. Die erste ist nahtlos in den Beleuchtungspart des obigen BTF Renderingalgorithmuses integriert und für statische Geometrien optimiert. Die zweite Methode behandelt dynamische Objekte. Dazu werden hardwarebeschleunigte *Occlusion Queries* verwendet, um die Sichtbarkeitsberechnung durchzuführen. Diese Methode ist einerseits simpel und leicht zu implementieren, anderseits ist sie schnell und produziert weniger Artefakte, als vergleichbare Methoden. Zusätzlich ist die Verwendung von veränderbarer, entfernter HDR Beleuchtung integriert.

Das menschliche Wahrnehmungssystem ist das eigentliche Ziel jeglicher Anwendung in der Computergrafik und kann daher selbst als Teil einer erweiterten *Rendering Pipeline* gesehen werden. Daher kann das Rendering selbst optimiert werden, wenn man die menschliche Wahrnehmung verschiedener visueller Aspekte der berechneten Bilder analysiert. Teil der vorliegenden Arbeit ist die Beschreibung eines Experimentes, das menschliche Schattenwahrnehmung untersucht, um das Rendern der Schatten zu beschleunigen.

Ein weiteres Teilgebiet der Kleidungsvisualisierung in der Computergrafik ist die Animation der Kleidung und von Avataren für Präsentationen. Diese Arbeit beschreibt zwei neue Methoden auf diesem Teilgebiet. Einmal ein Algorithmus, der für die automatische Generierung neuer Animationssequenzen verwendet werden kann und zum anderen einen Kompressionsalgorithmus für eben diese Sequenzen.

Die automatische Generierung von völlig neuen, anpassbaren Animationen basiert auf dem Konzept der Ähnlichkeitssuche. Hierbei werden die einzelnen Schritte von gegebenen Basisanimationen auf Ähnlichkeiten hin untersucht, die zum Beispiel die Geschwindigkeiten einzelner Objektteile sein können. Die Identifizierung dieser Ähnlichkeiten erlaubt dann Sprünge innerhalb der Basissequenz, die dazu benutzt werden können, endlose, neue Sequenzen zu erzeugen.

Die Übertragung von animierten 3D Daten über bandbreitenlimitierte Kanäle wie ausgedehnte Netzwerke, Mobilfunk oder zu sogenannten *thin clients* erfordert eine effiziente Komprimierung. Die zweite, in dieser Arbeit vorgestellte Methode, ist ein Kompressionsschema für Geometriedaten. Ähnlich wie bei der Kompression von BTF Daten wird die PCA in Verbindung mit Clustering benutzt, um die animierte Geometrie zu analysieren und in sich ähnlich bewegende Teile zu segmentieren. Diese erkannten Segmente lassen sich dann hoch komprimieren. Der Algorithmus arbeitet automatisch und erlaubt zudem eine sehr exakte Rekonstruktionsqualität nach der Dekomprimierung.

Contents

CHAPTER 1

Preface

1.1 Motivation

Visualization of real-world surface materials is a broad research area in computer graphics. Apart from the movie and entertainment industry, computer generated images containing material surfaces are also of interest for virtual-reality applications, e-commerce, customer decision support and interior and fashion design.

One of the main goals of these images is to convince the viewer of the realism of the shown objects. Besides the correct modeling of the geometry, the used surface reflection properties contribute an essential part to the overall acceptance of the image. This of course has to be combined with realistic illumination conditions and the resulting shadows. For some applications, the objects have to be realistically animated as well. For internet applications, this data has to be efficiently transferred over bandwidth limited channels. Therefore, it is indispensable to acquire, store and render the reflection properties fast and efficiently.

A specific example of the areas described above, is the realistic visualization of virtual humans, especially the visualization of textiles and clothing. Here, besides the correct physically-based simulation of the geometry, methods to compute correct self-shadowing and efficiently handle material reflection properties are needed. Without these methods, virtual try on applications and preservation of the *look & feel* of the cloth are not possible.

Computation of the cloth geometry and shape consists of solving physically-based equations to determine new vertex positions relative to a base mesh. Therewith, effects like friction and inherent material forces can be simulated. Also draping and collision detection, especially self-collision, has to be considered. As of today, there exist a lot of methods to efficiently compute these new vertex positions, e.g. incorporating measured physical material properties, even on the GPU (graphic processing unit).

Concerning the cloth surface, in addition to the microstructure, the mesostructure of a fabric is of great importance for the reflectance behavior of cloth. The mesostructure is responsible for fine-scale shadows, occlusions, secularities and subsurface scattering effects. Altogether these effects are responsible for the *look & feel* of cloth. There are essentially two techniques of cloth rendering according to the way in which mesostructure is captured.

The first approach explicitly models the mesostructure of the fabric in detail and renders it using different lighting models and rendering techniques. This approach requires a great amount of modeling and user intervention in combination with sophisticated geometry models.

The second approach, which is used in this thesis, is based on the explicit measurement of the optical surface reflection properties using camera devices. To store the accrued data, the bidirectional texture function (BTF) has been proved valuable. This representation captures the spatial varying reflection properties under different viewing and lighting conditions. To achieve a certain level of realism, the huge data amount has to be stored and accessed efficiently, to allow for interactive rendering. Therefore, this thesis includes a chapter for BTF compression and rendering and introduces state-of-the-art methods.

To further enhance the realism of virtual cloth, the addition of some sort of time-dependent motion is beneficial. This is usually done via animated virtual characters combined with natural effects, like for example wind and gravity.

Some areas of usage also require the transmission of the animation data over bandwidth limited channels, for example the internet or cellular phones. Therefore, efficient compression schemes are needed. For presentations it is also desired to automatically generate new animation sequences, that is without the intervention of a human animator.

When displaying real-world objects under real-world illumination conditions, computer graphics always tries to make it *look right* to the human observer. For non-artificial objects this is most difficult, because the human observer is used to see the objects in the real-world, and concerning structure and material surface reflection properties, is very sensitive to errors. Besides this, computer graphics has also to cope with low-dynamic-output devices in contrast to the human eye. On the other hand, slight variances in shadows and objects in motion are very good interpolated by the human brain. Therefore, using knowledge about the human visual system allows for approaches, that can speedup certain part of the rendering pipeline.

Combing all aspects mentioned above, an exemplary production work flow could look like the chart, shown in Figure 1.1.

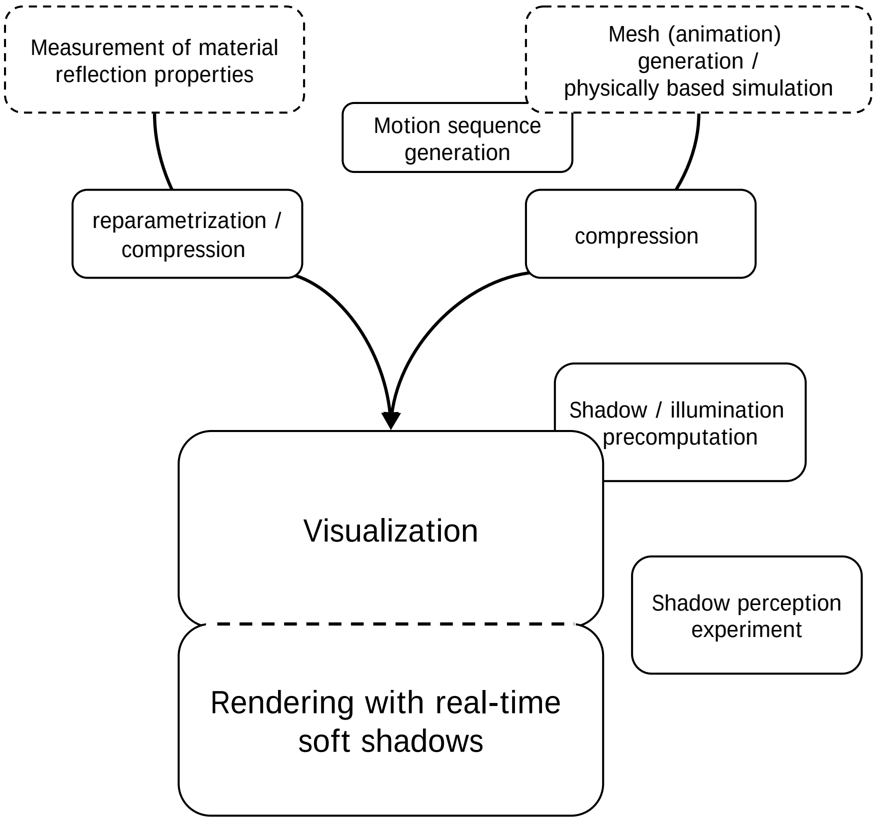

Measurement of material reflection properties
Mesh (animation) generation / physically based simulation
Motion sequence generation
reparametrization / compression
compression
Shadow / illumination precomputation
Visualization
Shadow perception experiment
Rendering with real-time soft shadows

Fig. 1.1: Overview chart of the aspects covered in this thesis (excluding dashed parts).

1.2 Main Contributions

Several aspects of the work described in this thesis have been already published at different conferences, journals and tutorial notes [Sattler *et al.* 2005a; Sattler *et al.* 2005b; Müller *et al.* 2005a; Wacker *et al.* 2004; Sattler *et al.* 2004a; Sattler *et al.* 2004b; Sattler *et al.* 2003; Hauth *et al.* 2002].

The content of this thesis is based on these publications, explaining the proposed methods in more detail and providing necessary background knowledge. This is completed with improvements and further results to the presented methods and algorithms.

The main contributions of this thesis can be summarized as follows:

- BTF compression scheme
- BTF rendering pipeline
- Animation sequence generation
- Animation sequence compression
- Enhanced BTF rendering pipeline with shadows
- Self-Shadowing for dynamic objects
- Evaluation of human shadow perception

The work presented in this thesis consists of the first practicable BTF compression scheme and the integration into the *Virtual Try-On* [VTO 2005] pipeline, including the efficient calculation of geometry self-shadowing. Furthermore, a visual perception experiment to evaluate human shadow perception is described. Concerning cloth animations, existing algorithms are adopted to animation sequence generation and a new animation sequence compression scheme is proposed. Most of the presented methods make use of programmable graphics hardware.

1.3 Thesis Overview

The rest of this thesis is organized according to the diagram shown in Figure 1.1.

In chapter 2 background and detailed information about *rendering techniques*, *geometry processing techniques*, *graphics hardware*, *shadows*, *animation*, *cloth visualization* and *data analysis techniques* is given.

Chapter 3 deals with the animation branch of the overview chart, including *sequence generation* and *sequence compression*.

Chapter 4 describes the shadow aspects of this thesis. This includes *perception* and *self-shadowing* for static and dynamic meshes.

Main subject of chapter 5 are material reflection properties. Here, emphasize is laid on *acquisition*, *compression* and *rendering* techniques.

After demonstrating the practical application of the described methods and approaches in chapter 6 on the basis of two industrial projects (*Virtual Try-On* and *Real-Reflect*), the thesis concludes in chapter 7 with a discussion and possible directions for future work.

Accompanying Video Material

Several described techniques in this thesis are accompanied by additional video material. The following icon indicates that a video file is available on the thesis DVD:

CHAPTER 2 Background

2.1 Rendering Techniques

The ultimate goal of nearly all computer graphic applications is to produce some kind of image on an output device. Therefore, some kind of computer understandable description of the content of the image has to exist. Due to the complexity of real-world physics and the deficiency of computer hardware, the way to generate realistic images is build upon approximations. The following section introduces basic and advanced topics of generating a realistic image in computer graphics and gives an overview of the physical background.

2.1.1 Radiometry

There exist mainly two different models to describe the physics of light. The first describes light as a flow of particles (*photons*) carrying energy, while the opposite describes light as an electromagnetic *wave*. *Radiometry* describes the entire radiant power and the quantities derived from it.

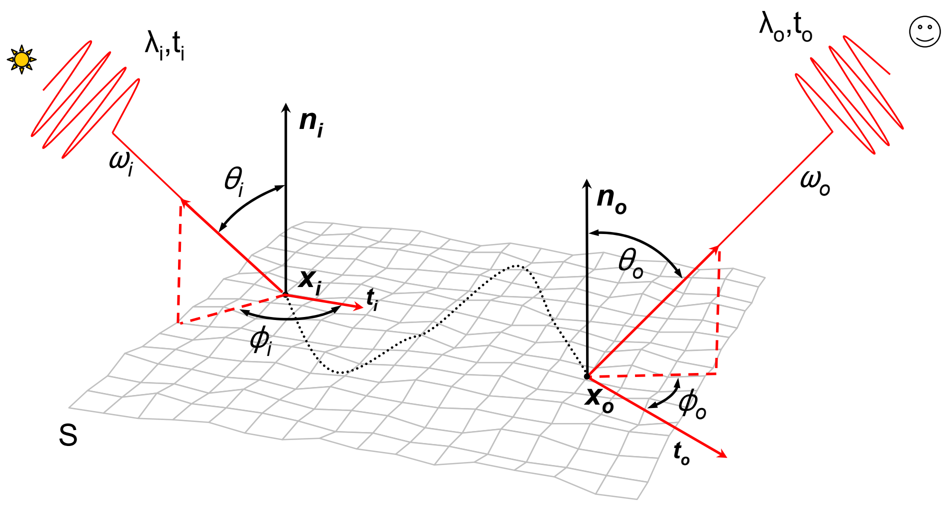


Fig. 2.1: General light-matter interaction.

Figure 2.1 illustrates the general light-matter interaction from a light source (sun) to an observer (face). A surface S is hit by a beam of light coming out of the direction ω_i with the wavelength λ_i at the time t_i. The surface is hit at the point $\mathbf{x_i}$ with the surface normal $\mathbf{n_i}$. The direction ω_i is defined by two angles, namely θ_i

and ϕ_i. The latter is attached to a local coordinate system vector $\mathbf{t_i}$.

After traveling through the matter, the beam leaves the surface at the time t_0 at the outgoing point $\mathbf{x_o}$ into the direction ω_o. It might has changed the wavelength to λ_o. Similar to the entrance direction, the direction is defined by the two angles θ_o and ϕ_o with the local coordinate vector $\mathbf{t_o}$. This description of the light-matter interaction therefore results in a 12-dimensional function.

For a more detailed description of optics, see for example [Bergmann & Schaefer 2004]. In the following, basic radiometric terms are introduced. The notation used is given in Table 2.1 and 2.2. All units are given in the *SI* (Système International d'unités [BIPM 2006]) system and the symbols are defined in [CIE1987].

symbol	radiant term	unit
ν	frequency	s^{-1}
λ	wavelength	m
E_{ph}	photon energy	J
Q_e	radiant energy	J
Φ_e	radiant flux	W
E_e	irradiance (incident)	Wm^{-2}
M_e	radiant exitance (outgoing)	Wm^{-2}
I_e	radiant intensity	Wsr^{-1}
L_e	radiance	$Wm^{-2}sr^{-1}$

Tab. 2.1: Important radiant terms.

symbol	term	value
h	Planck´s constant	$6.626 \times 10^{-34} Js$
c_0	speed of light	299 792 458 ms^{-1}

Tab. 2.2: Important constant values in radiometry.

A photon, as the atomic unit, is a quantum of light. It has a position, direction and a wavelength. Depending on the refractive index n of the medium through which it travels, it has a constant speed c:

$$\nu = \frac{c}{\lambda} \tag{2.1}$$

and a certain amount of energy:

$$E_{ph} = h\nu = \frac{hc}{\lambda} \tag{2.2}$$

The *radiant energy* Q_e, that is the energy sum of a number of photons over all wavelengths, per time gives the *radiant flux*:

$$\Phi_e = \frac{dQ_e}{dt} \tag{2.3}$$

with the unit *Watt [W]* with $W = Js^{-1}$.

Radiant flux area density, that is the differential flux of light, which hits or leaves a differential area at a surface point $\mathbf{x}$ is divided into the

irradiance (incident):

$$E_e(\mathbf{x}) = \frac{d\Phi_e}{dA} = \int_{2\pi sr} L_e \cdot cos\theta \cdot d\Omega \tag{2.4}$$

and the *radiant exitance* (outgoing) or as it is also known in computer graphics *radiosity* :

$$M_e(\mathbf{x}) = \frac{d\Phi_e}{dA} = \int_{2\pi sr} L_e \cdot cos\theta \cdot d\Omega \tag{2.5}$$

The *solid angle* Ω is measured in *steradians* $[sr^{-1}]$. It is defined as the solid angle that, having its vertex at the center of a sphere, cuts off an area of the surface of the sphere equal to that of a square with sides of length equal to the radius of the sphere:

$$d\Omega = \frac{dA}{r^2} \tag{2.6}$$

The *radiant intensity* of a light source in a given direction $\vec{\omega}$ is defined as follows:

$$I_e(\vec{\omega}) = \frac{d\Phi_e}{d\Omega} \tag{2.7}$$

One of the most important radiometric terms in computer graphics is *radiance* (see also Figure 2.2). That is the radiant flux transmitted by an elementary beam passing through a given point **x** and propagating in the solid angle $d\Omega$ containing the outgoing direction $\vec{\omega}$. dA is the area of a section of that beam and θ defines the surface normal at point **x**:

$$L_e(\mathbf{x},\vec{\omega}) = \frac{d^2\Phi_e}{dA \cdot cos\theta \cdot d\Omega} = \frac{dI_e}{dA \cdot cos\theta} \tag{2.8}$$

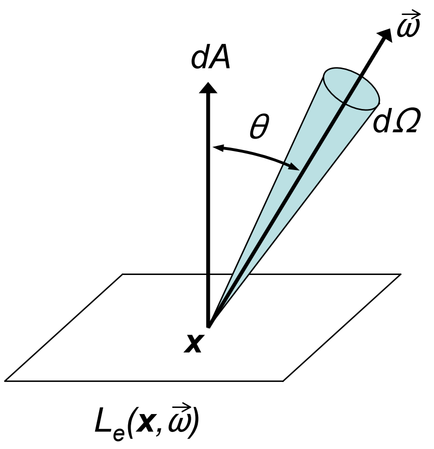


Fig. 2.2: Basic principle for radiance.

2.1.2 Colorimetry & Photometry

In contrast to radiometry (see Section 2.1.1), which is the science of the physical measurement of electromagnetic energy, *colorimetry* is the science that describes colors independently of the observer. To include the properties of the human eye and to provide psychophysical measurements, *photometry* is used. Here, for each unit used in radiometry, a counterpart exists.

Colorimetry

The CIE (Commission Internationale de l'Éclairage) [CIE 2006] has defined a standard observer and a set of guidelines for performing color measurements. Also, standard light sources, such as the *D50* or *D65* are defined [CIE 2004]. The D65 standard resembles natural daylight including the ultraviolet region. In [CIE 2004] also color matching and color spaces such as *CIE-LAB*, *CIE- LUV* or *CIE-XYZ* are defined.

Photometry

The human eye is sensitive to electromagnetic radiation with wavelengths between 380 and 770 nm. It is a complex and nonlinear detector using two types of photoreceptors (*cones* and *rods*). Nonlinearity is involved, because the sensitivity varies with the wavelength. An average human visual response function is the *spectral luminous efficiency function* $V(\lambda)$:

$$V(\lambda) = \frac{\Phi_e(\lambda_m)}{\Phi_e(\lambda)} \tag{2.9}$$

Details and data values can be found in [CIE1987]. Two important details are eminent. The sensitivity varies also with overall brightness which is perceived. That is, during daylight, the *photopic* curve is valid, while during the night, the *scotopic* curve is valid. During the day, there is a clear peak in sensitivity around 550 nm, which is perceived as *green*.

2.1.3 Shape

A real-world object has two main properties: *shape & material*. These two properties may also vary over *time* and may be influenced by the *environment* as shown in Figure 2.3.

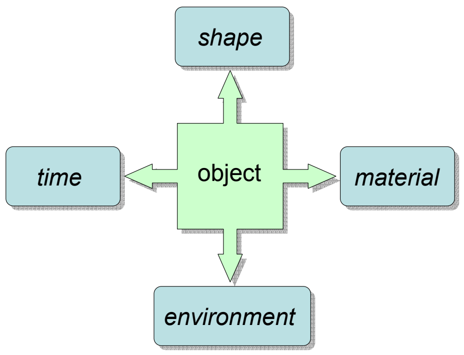


Fig. 2.3: Object representation.

Figure 2.4 shows the basic representation of *shape* or *geometry* in computer graphics. Starting on the left side, an object is first segmented into basic surfaces. These surfaces are substituted by a polygonal representation. A polygon finally is described by (corner-) *vertices*, which are connected by *edges*. Therefore, object animation (see Section 2.5), that is, the *time* aspect, can simply be achieved by changing vertex positions in space over time. While there exist other (mathematical inspired) non-polygonal representations such as parametric or quadric surfaces [Foley *et al.* 1996], modern graphics hardware is optimized to handle triangles (see Section 2.3).

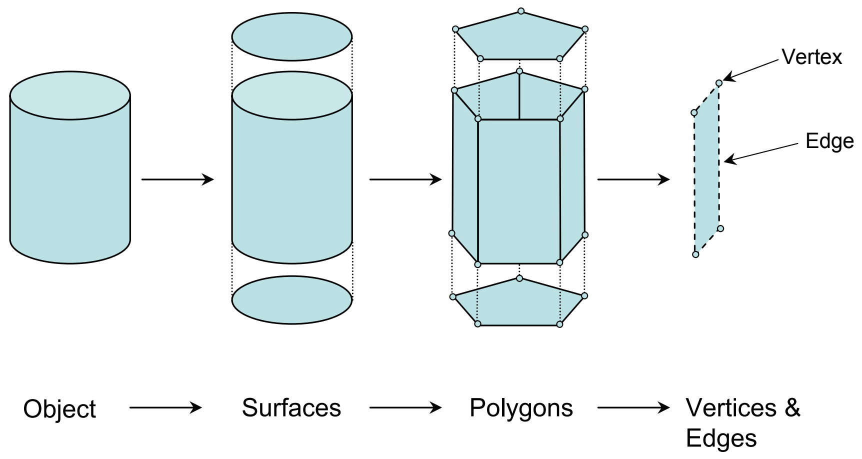


Fig. 2.4: Basic geometry representation.

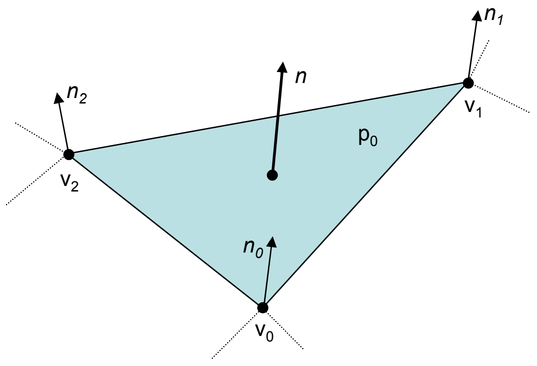


Fig. 2.5: Triangle setup.

Figure 2.5 shows the basic triangle setup. As the simplest polygonal geometric primitive to describe an area in space, a triangle p_0 is defined by its corner vertices $v_0 - v_2$. It is customary, to use the right hand rule to define the *front facing* side. The face normal n is perpendicular to the triangle face. This conventions will later be used for lighting calculations and culling algorithms.

2.1.4 Material

In the following the term *material* or *material properties* means all physical properties like reflection, transparency, specific gravity and so on. In the context of computer graphics, especially the optical properties are important. Table 2.3 lists important terms, their dimension and the section where they are described in detail.

term	*dimension*	*description*	*section*
BSSRDF	8D	Bidirectional Surface Scattering Reflectance Distribution Function	2.1.5
BTF	6D	Bidirectional Texture Function	2.1.6
BRDF	4D	Bidirectional Reflectance Distribution Function	2.1.7
RF	4D	Reflection Fields	2.1.7
LF	4D	Surface Light Fields	2.1.7
DSRF	4D	Diffuse Subsurface Reflectance Function	2.1.7
displacement map	2D	image storing additional geometry information	2.1.8
bump map	2D	image storing normal information	2.1.8
texture	2D	image storing color information	2.1.8

Tab. 2.3: Important terms in rendering.

While the general scattering of light is a 12-dimensional process, it is common to make the following assumptions to reduce the complexity (see also Figure 2.1):

- light transport take zero time (t_i=t_o), i.e. no phosphorescence;
- reflectance behavior of the surface is time invariant ($t_0 = t_i = t_o$)
- no change in wavelength (λ_i=λ_o), i.e. no fluorescence;
- wavelength is not continuous, but discretized into red, green and blue bands ($\lambda \rightarrow \lambda_{RGB}$);

Now, the 12 dimensions are reduced to 8 and the function is called *BSSRDF* (Bidirectional Surface Scattering Reflectance Distribution Function) [Nicodemus *et al.* 1977].

In computer graphics light is modeled as *ray optics*, that is, light interaction is treated as independent rays traveling through space. While this geometrically approach might be computational efficient, complex effects of light, like polarization, interference or diffraction are generally also neglected.

Figure 2.6 gives shows an overview of reflectance functions, their dimensions and their connections to each other.

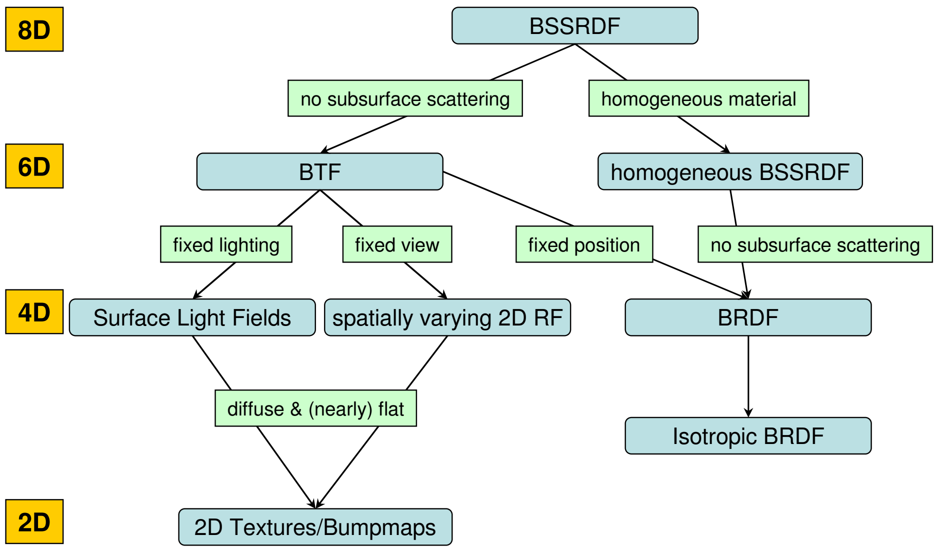


Fig. 2.6: Global overview of reflectance functions.

2.1.5 8D: BSSRDF

A practical model for rendering of the BSSRDF has been proposed by Jensen et al. [2001] and is based on a dipole approximation of a diffusion model. The model handles homogeneous materials via two parameters obtained from a single HDR image (σ_a: absorption cross section and σ'_s: reduced scattering cross section). Visual good results were obtained for materials like marble and fluids like milk.

Goesele et al. [2004] presented a laser-based measurement setup for translucent inhomogeneous objects. They assumed diffuse surface reflection in combination with a strong subsurface scattering, because no angular dependency is measured.

2.1.6 6D: BTF

If no implicit subsurface scattering is considered, the BSSRDF reduces to the six-dimensional BTF (bidirectional texture function), which mainly will be used in the latter methods introduced in this thesis. First proposed by Dischler [1998] and Dana et al. [1999b], the BTF still captures the most important visual effects of nearly flat material surfaces. This includes shadowing, masking, and self-interreflections (see Figure 2.7).

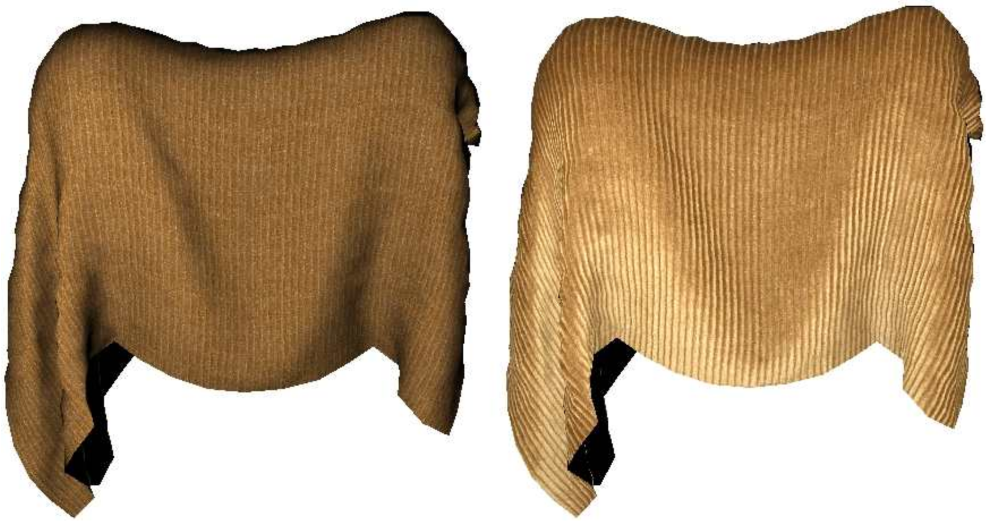


Fig. 2.7: Comparing simple texture mapping and rendering from a measured BTF.

The BTF might also be interpreted as a ABRDF (apparent bidirectional distribution function). As explained in 2.1.7, the BRDF is the reflection function on an infinitesimal surface element. Extending this local function spatially, as shown in Figure 2.8, gives a special BRDF for every surface point. This ABRDF contains parts of the material BRDFS, but also includes all shadowing and masking effects described above.

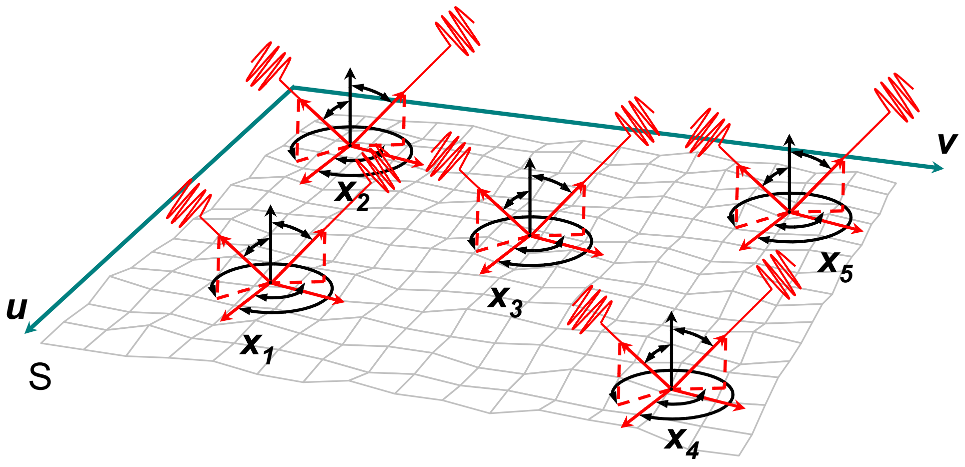


Fig. 2.8: BTF representation consisting out of spatial distributed ABRDFs.

2.1.7 4D: BRDF / LF / RF

BRDF

For many applications it is convenient to drop spatial dependence and consider reflection taking place on an infinitesimal surface element. This process is described by the 4-dimensional BRDF (see Figure 2.9) and the classical measurement device for this quantity is the *gonioreflectometer*, which samples the angular dependency sequentially by positioning a light source and a detector at various directions from the sample [Nicodemus *et al.* 1977]. Several methods attempted to reduce measurement times by exploiting CCD-chips for taking several BRDF-samples at once. Ward [Larson 1992] used a hemispherical half-silvered mirror and a camera with a fish-eye lens to acquire the whole exitant hemisphere of the flat probe at once. Alternatively one could take images from a curved sample as it was done by Marschner et al. [Marschner *et al.* 1999] and Matusik et al. [Matusik *et al.* 2003]. The latter work was tailored to measuring isotropic BRDFs. It demonstrates also how measurement times can be significantly reduced by using sophisticated learning algorithms and a database of densely acquired BRDFs.

LF / RF

Pure image-based rendering considers the flow of light independent of a physical surface and became popular with the works on light fields (LF) [Levoy & Hanrahan 1996] and lumigraphs [Gortler *et al.* 1996]. They observed that the five dimensional *plenoptic function* (pencil of light rays flowing through points in

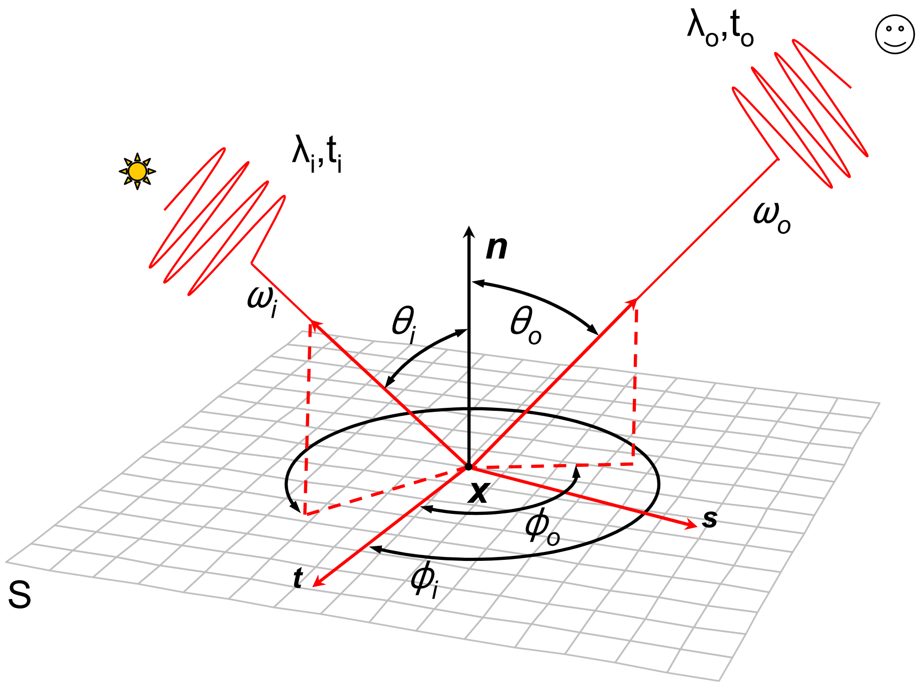


Fig. 2.9: BRDF representation for an infinitesimal surface element.

space) can be described by a 4-D function if the viewer moves in unoccluded space outside or inside a virtual surface (e.g. a cube) over which the light field is parameterized. The variation of this light field according to a 4-D light field of radiance incident at the virtual surface is described by the 8-D *reflectance field*. [Debevec *et al.* 2000]. If parameterized over a physical surface this structure is equivalent to the BSSRDF. Fixing the incident light field gives the light field as originally introduced by Levoy et al. [1996] and Gortler et al. [1996] and which has been sampled using an array of cameras. Parameterized over physical surfaces it is called the surface light field (SLF) [Miller *et al.* 1998; Wood *et al.* 2000]. The surface light field is measured using many images from different viewpoints of an object with known geometry.
In order to capture the lighting variability of the reflectance field, many images of the scene under varying lighting conditions have to be taken. Debevec et al. [2000] built a so called *light-stage* which records the appearance of a human face while a light source is rotating around the face. They assumed infinitely distant lighting and fixed the view, ending up with spatially varying 2-D *reflectance functions*(RF) to reduce the dimensionality of the reflectance field. For measuring the reflectance

functions of small and nearly planar objects Malzbender et al. [2001] constructed a hemispherical gantry attached with 50 strobe light sources. They also introduced *Polynomial Texture Maps* (PTM), a compact representation for the acquired data that is especially suited for diffuse materials. A very complex acquisition setup was built by Matusik et al. [2002]. It captures the object also from varying viewpoints and handles objects with complex silhouettes using multi-background matting techniques and thus actually (sparsely) samples a 6D slice of the reflectance field. Masselus et al. [Masselus *et al.* 2003] fixed the viewpoint again but instead used a spatially located light basis which enabled the relighting of the scene by the full 4-D incident light field.

2.1.8 2D: Texture, Bump & Displacement maps

Texturing

After the rasterization stage (see Section 2.35), a triangle might occupy several pixels on the screen. There exist several ways to assign a color value to a pixel. With basic shading models only a uniform or interpolated color value is assigned to each pixel.

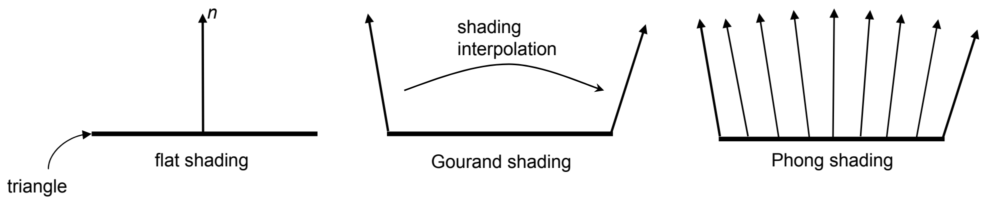


Fig. 2.10: Different shading models.

Figure 2.10 shows several basic shading models. Beginning on the left, *flat shading* assigns one color per polygon, while *Gourand shading* interpolates the colors computed at the vertices over the polygon. The more sophisticated *Phong shading* interpolates the normals of the polygon and then computes the per pixel shading.

To resemble real-world surfaces, a two-dimensional *texture* can be used. That is, some kind of image is mapped onto the triangle or a set of triangles. The image can be a part of a real-world object, hand-drawn or computer generated (see Figure 2.11).
The image lives in the texture space (u, v) which is parameterized in $[0, 1]$. A *texture coordinate* (u_n, v_n) is assigned to each vertex $(v_0 - v_2)$ of a specific triangle.

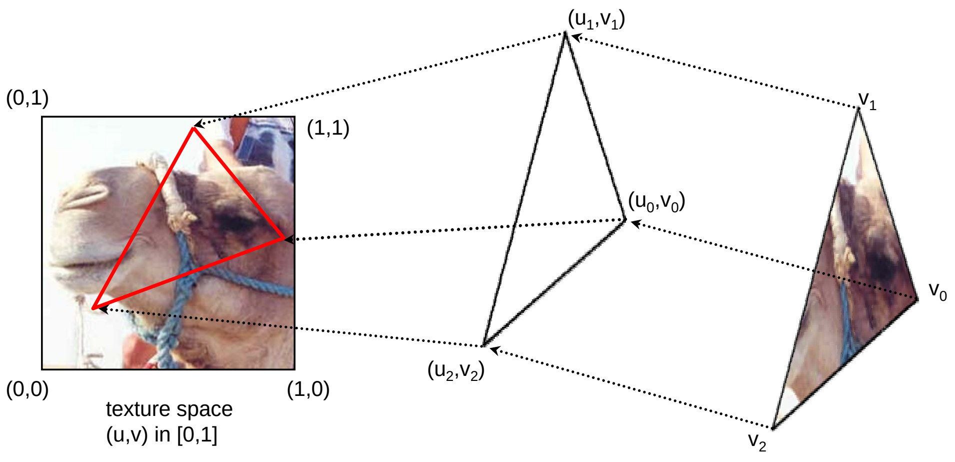


Fig. 2.11: Basic texturing principle.

Bump Mapping

In contrast to this, *Bump mapping* can be used to generate pseudo micro-geometry by perturbing the normals to generate the shading effects of virtual geometry [Blinn 1978] (see Figure 2.12). The effect is achieved at the point, where the lighting calculation is done. Bump mapping can also be combined with normal texturing in a multi-pass texturing process.

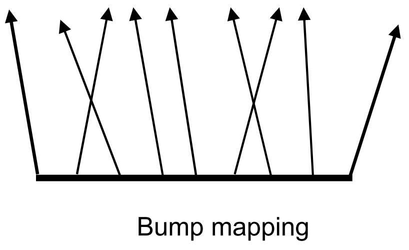


Fig. 2.12: Bump mapping with disturbed per-pixel normals.

Displacement Mapping

Another kind of information which can be stored in 2D images is *displacement* as introduced by Cook [Cook 1984]. Vertex positions of a base geometry are shifted according to movement vector information stored in a displacement map as shown in Figure 2.13. It is also possible to generate complete new geometry for example through subdivision (see Section 2.2.1), if the displacement map has a higher spatial resolution then the base geometry. Displacement mapping is hardware supported in several ways [Pixar 2005; Matrox 2006] and also included in Shader Model 3.0 (see also Section 2.3).

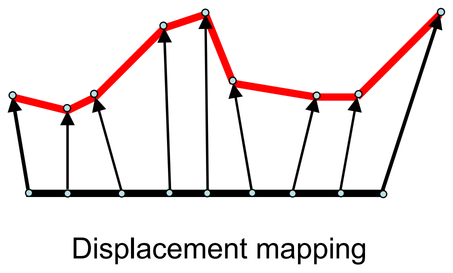


Fig. 2.13: Displacement mapping with shifted vertex positions.

2.1.9 Data Acquisition

There exist two orthogonal approaches to generate reflection data. The first is the explicit modeling, the second is the measurement. Because this thesis is based on the latter, modeling is only briefly mentioned in the following, while the rest of the section is fully dedicated to the measurement process.

Modeling

There are essentially two techniques of cloth rendering according to the way in which mesostructure is captured. The first approach explicitly models the mesostructure of the fabric in detail and renders it using different lighting models and rendering techniques [Gröller *et al.* 1995; Daubert & Seidel 2002]. Although these algorithms produce impressive results and some of them are already applicable at interactive frame rates, using these methods, it is difficult to reproduce the special appearance of a given fabric.

BTF-Measurement

From now on, the BTF can be seen as a measured six-dimensional slice of the general light scattering function of a surface S:

$$\mathbf{BTF}_{rgb}(\mathbf{x}, \theta_i, \phi_i, \theta_r, \phi_r) := \int_S \mathbf{BSSRDF}_{rgb}(\mathbf{x}_i, \mathbf{x}, \theta_i, \phi_i, \theta_r, \phi_r) d\mathbf{x}_i$$

It this sense the BTF integrates subsurface scattering from neighboring surface locations, as it is done by the most existing measurement setups. Nevertheless, this definition allows also for the mathematical interpretation of the BTF as spatially varying BRDF. The corresponding previous work can be grouped roughly into two categories.

The first group captures the geometry of a small object and its reflection properties parameterized over the surface, that is the spatially varying BRDF. The works of Lensch et al. [2001] and Furukawa et al. [2002] fall into this category. They capture the geometry of the object using laser scanners and take several images under varying light and viewing conditions. The methods differ in their data representation. While Furukawa et al. map the images onto the triangles of the model and compress the appropriately reparameterized data using tensor product expansion, Lensch et al. fitted the data to a parametric reflection model. In order to cope with insufficient sample density they used an iterative clustering procedure.

The second group aims at capturing the appearance of an opaque material independently from geometry. These methods have in common, that they capture the BTF of a planar material sample. The acquired data can be used instead of simple 2D-textures and mapped onto arbitrary geometry. In the following these methods are described in detail.

Gonioreflectometer-like Setup with CCD-Chips

The most common approaches use a gonioreflectometer-like setup with a CCD-chip instead of a spectrometer in order to capture the spatial variation of reflection. This approach has proved to be reliable and several variations of it have been published. However its drawback are the long measurement times.

The first measurement system that used such a gonioreflectometer like setup as depicted in Figure 2.14 was presented in the pioneering work of Dana et al. [1999b]. Their system takes 205 images of isotropic materials which is a too sparse sampling for high-quality rendering, in particular for rough surfaces and materials with

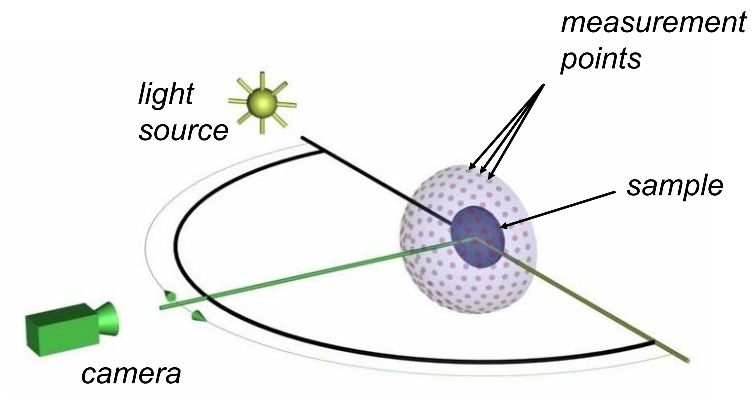


Fig. 2.14: Capturing the BTF of a planar sample using a gonioreflectometer-like setup with a fixed light source, sampleholder and a moving camera.

strong specular pikes. Even though they mentioned the possibility of using the data for rendering, the original intent of the work was building up a material database for computer vision related tasks such as texture recognition, texture segmentation and shape-from-texture. They measured 61 real-world surfaces and made them available through the CUReT database [Curet 2005].

Similar, but improved versions of the measuring system were described in [McAllister *et al.* 2002] and [Hauth *et al.* 2002]. Some measurements of the latter system are now also publicly available through the BTF database Bonn [BTFDBB 2005]. This system will be described in greater detail in the following.

Measurement Setup The measurement setup is designed to conduct an automatic measurement of a BTF that also allows the automatic alignment and post-processing of the captured data. A high-end digital still camera is used as image sensor. The complete setup, especially all metallic parts of the robot, are covered with black cloth or matte paint, with strong diffuse scattering characteristics.

The system uses planar samples with a maximum size of $10 \times 10\ cm^2$. In spite of these restrictions, measurement of a lot of different material types, for example fabrics, wallpapers, tiles and even car interior materials is possible. As shown in Figure 2.15, the laboratory consists of a HMI (Hydrargyrum Medium Arc Length Iodide) bulb, a robot holding the sample and a rail-mounted CCD camera (Kodak DCS Pro 14N). Table 2.4 shows the sampling density of the upper hemisphere for light and view direction which results in $n = 81$ unique directions for camera and light position. Hence, 6561 pictures of a sample are taken.

Video I

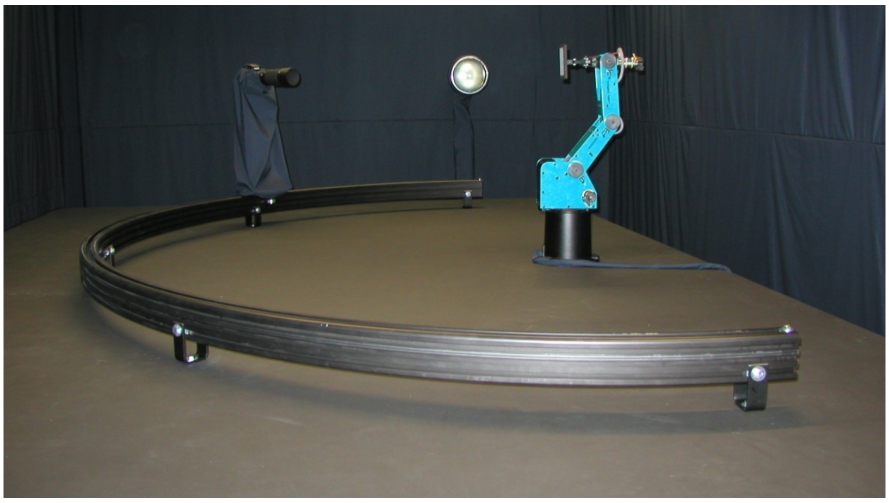


Fig. 2.15: Measurement setup of the Bonn-System consisting out of an HMI lamp, a CCD camera and a robot with a sample holder.

θ [°]	$\Delta\phi$ [°]	No. of images
0	$-^*$	1
15	60	6
30	30	12
45	20	18
60	18	20
75	15	24

Tab. 2.4: Sampling of viewing and illumination angles of the BTF database Bonn. *= only one image taken at $\phi = 0°$

Figure 2.16 shows several measured samples. The top row shows frontal views of the different samples, whereas the bottom row shows oblique views. In the latter case especially the mesostructure of the samples becomes visible. Each raw image is 12 megabytes in size (lossless compression) with a resolution of 4500×3000 pixels (Kodak DCR 12-bit RGB format). A film is also available, showing the complete measurement process.

With this setup, the measuring time is about 14 hours, where most of the time is needed for the data transfer from the camera to the host computer.

Calibration To achieve high-quality measurements, the equipment has to be calibrated.

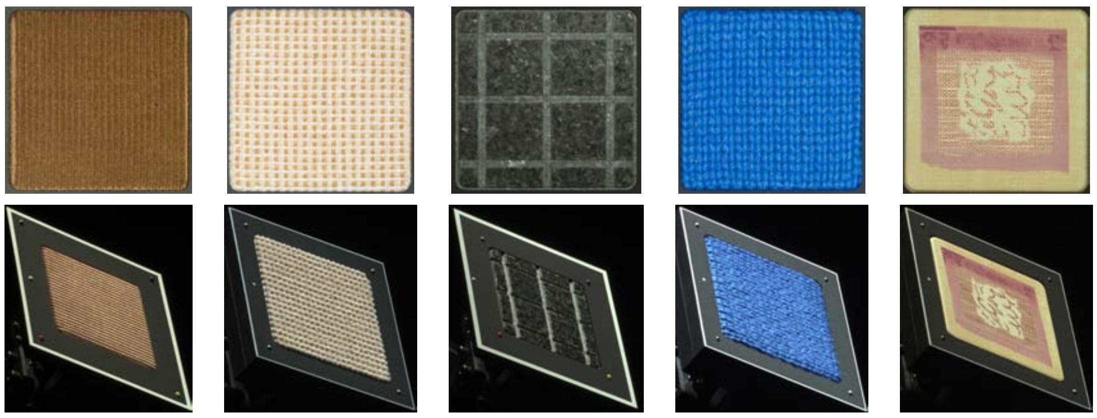


Fig. 2.16: Measured BTF samples; from left to right (top row): CORDUROY, PROPOSTE, STONE, WOOL and WALLPAPER. Bottom row: perspective views $(\theta = 60, \phi = 144)$ of the material and sample holder with illumination from $(\theta = 60, \phi = 18)$. Note the mesostructure and changing colors.

- To compensate the positioning error due to the robot and the rail system, one has to track the sample holder mounted on the robot arm using the camera. Experiments determined that these errors are small in the described setup. Therefore, marker points, which are placed on the sample holder, are detected only during the post-processing phase, allowing a software jitter correction.

- A geometric calibration has to be applied to the camera to reduce geometric distortion, caused by the optical system of the camera. Details and further references to camera calibration can be found for example in [Zhang 2000].

- For each sample to be measured, the aperture of the camera is adjusted in such a way that the number of saturated or dark pixels in the pictures is minimized given a fixed aperture during the measurement process.

- To achieve the best possible color reproduction, the combination of the camera and the light source has to be color calibrated. For the measurement of the camera color profile a special CCD-Camera standard card (Gretag Macbeth - Color Checker DC) is used.

Data Postprocessing After the measurement the raw image data is converted into a set of rectified, registered images capturing the appearance of the material for varying light and view directions. Now, a complete set of discrete reflectance values for all measured light and viewing directions can be assigned to each texel of a 2D texture.

Registration is done by projecting all sample images onto the plane which is defined by the frontal view ($\theta = 0, \phi = 0$). To be able to conduct an automatic registration, borderline markers were attached to the sample holder plate, see Figure 2.16. After converting a copy of the raw data to a binary image, standard image processing tools are used to detect the markers. In the following steps the mapping (which maps these markers to the position of the markers in the frontal view) is computed and utilized to fill the registered image with appropriate colors.

To convert the 12-bit RGB images stored in the proprietary format of the camera manufacturer to standard 8 bit RGB file formats, the standard color profiles provided with the Camera SDK (look and output profile) and camera (tone curve profile) are applied to the image. The most appropriate 8 bit color range is extracted after applying an exposure gain to the converted data.

After this postprocessing step, the final textures are cut out of the raw reprojected images and resized appropriately (256×256 pixels in size for probes in the database, up to about 800×800 in principle). A final dataset with 256×256 pixels spatial resolution has a data amount of 1.2GB.

Using Video Cameras

Koudelka et al. [2003] presented a system resembling the before mentioned ones, but it fixes the image sensor (a Canon XL-1 digital video camera) and moves the light source (a white LED mounted on a robot arm). The employed hemisphere sampling results in a dataset of about 10.000 images. Due to the use of a video camera with relatively low resolution compared to a high-end still camera, a measurement takes about 10 hours. Samples from this system are publicly available for research purposes and include interesting natural materials like lichen or moss and man-made materials like carpet or even LEGO$^{\text{TM}}$bricks.

Using Mirrors

Inspired by BRDF measurement techniques, it has also been proposed to use mirrors for BTF measurement in order to avoid hemispherical movements or to make several measurements in parallel.

An approach using a concave parabolic mirror has been published by Dana and Wang [2004]. In this device the parabolic mirror is focused on the sample and thus an image of the mirror captures the appearance of the surface point in focus as seen from different viewing directions. Spatial variation of the sample is captured by planar movement of the mirror (or the sample). Illumination from different directions is achieved by focusing a light beam on the appropriate spot in the mirror. With this setup no hemispherical movement is required and the resulting data is of high quality. But a high spatial resolution (comparable to the gonioreflectometer-like devices which achieve about 300DPI) requires an enormous amount of images. Please note also that interreflections and subsurface scattering from neighboring parts of the surface are not integrated. Furthermore for a substantial range of azimuthal angles ϕ the covered range of the polar angle θ is restricted by the size of the mirror. For example for $\phi = \pi$ the presented prototype can capture polar angles only up to $22.6°$.

Han et al. [2003] presented a measurement system based on a kaleidoscope which allows to capture several images of the whole sample at once. The advantages in measurement time and registration precision (no moving parts) are accompanied by a number of disadvantages. Multiple reflections on mirrors (not perfect reflectors) cause low image quality and lead to a difficult color calibration. Slight asymmetries in the configuration of the mirrors result in registration errors. Angular sampling and spatial resolution are often coupled, that is a higher angular resolution leads to a lower spatial resolution.

Radloff [2004] has analyzed different kaleidoscope configurations by simulation and built several prototypes. But due to the mentioned difficulties in building a perfect kaleidoscope the quality of the results turned out to be rather low.

Using a camera array

For a fast high quality acquisition of BTFs, an array of 151 digital still cameras mounted on a hemispherical gantry is proposed in [Müller *et al.* 2004a]. Figure 2.17 shows a sketch and a real image of the DOME device. Details can be found in [Müller *et al.* 2005b].

A similar gantry with mounted light sources was used by Malzbender et al. [2001] to capture polynomial texture maps (PTMs). Although the setup is costly to build, a camera array is capable of measuring many samples in a short time. Due to the parallel structure of the acquisition, the example setup would be capable of capturing a BTF dataset of $151^2 = 22801$ images in less than one hour. No moving parts are needed. Therefore, the region of interest (ROI) is known for every camera and can be extracted at subpixel precision. Hence, there is no need for a time-consuming detection of the ROI, the post-processing (reprojection, geome-

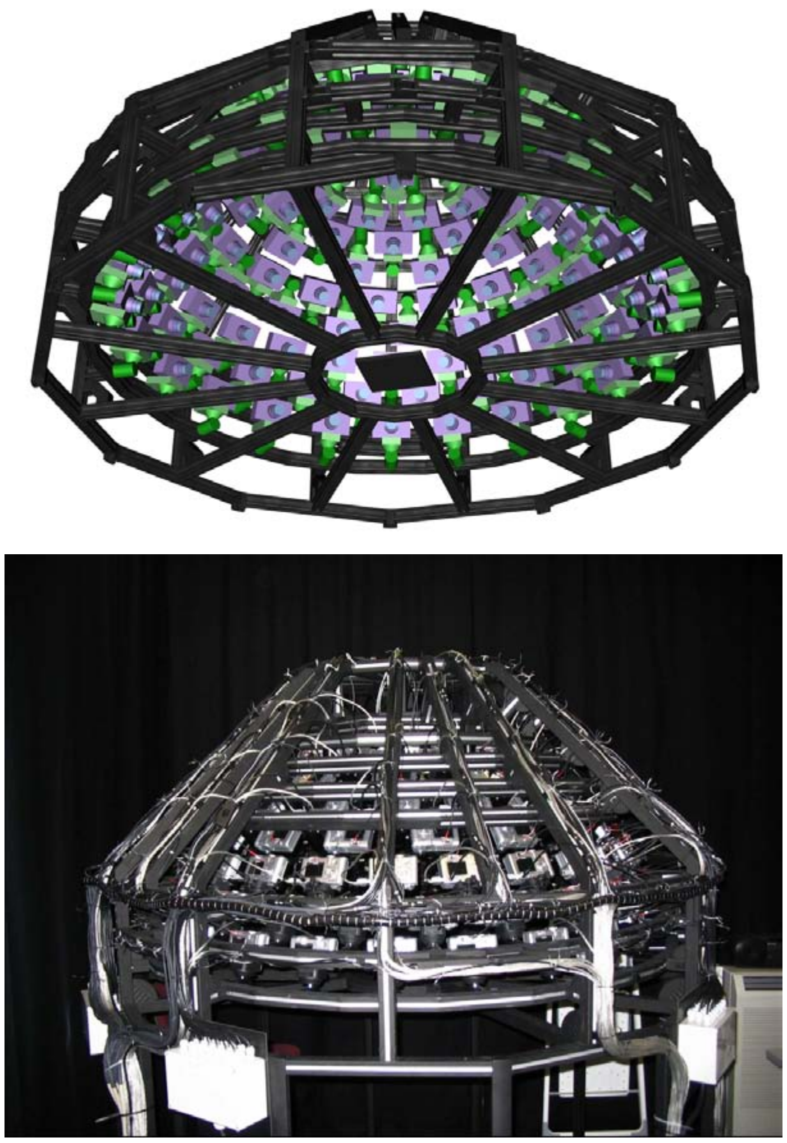


Fig. 2.17: Sketch and real image of the proposed camera array. 151 digital cameras with built-in flash lights are mounted on a gantry, focusing on the sample, which is placed in the center of the hemisphere.

tric correction, color correction) is fast enough to be done in parallel to the measurement. The angular resolution depends on the number of cameras and the spatial resolution on the imaging chips. This will result in a high angular resolution; every measured direction represents an average solid angle of only 0.04161 steradians. The spatial resolution would be up to 280DPI for a resulting BTF texture size of

1024x1024 pixels. As light sources, the built-in flash lights of the cameras will be used.

Discussion

Currently only the standard gonioreflectometer-like measurement setups have proven that they can be used to capture high-quality BTFs reliably. Their drawback is the speed - several hours is too long and makes measured BTFs an expensive resource. Using mirrors may be a promising approach in the future, but the current systems are far from reaching the resolution and quality of the gonioreflectometer-like setups. Using a camera array will greatly reduce measurement times while keeping quality and resolution at the expense of the costs for a large number of cameras.

2.1.10 Compression

Due to its huge size the pure image-based representation of a BTF consisting of the thousands of images taken during the measurement process is neither suitable for rendering nor for synthesis. In order to achieve real-time frame rates and acceptable synthesis times, some sort of data-compression has to be applied.

Such a method should of course preserve as much of the relevant features of the BTF as possible, but should also exploit the redundancy in the data in an efficient way and provide a fast, preferably real-time decompression stage. An optimal method would achieve high compression rates with low error and real-time decompression. For integration into current real-time rendering systems an implementation of the decompression stage on modern GPUs would also be of great value.

Most existing compression techniques interpret the BTF as shown in Figure 2.18: as a collection of discrete textures

$$\left\{T_{(\mathbf{v},\mathbf{l})}\right\}_{(\mathbf{v},\mathbf{l})\in\mathcal{M}},$$

where $\mathcal{M}$ denotes the discrete set of measured view- and light-directions, or as a set of spatially varying *apparent* BRDFs (ABRDF, the term was introduced in a paper of Wong et al. [1997]):

$$\{B_{\mathbf{x}}\}_{\mathbf{x}\in I\subset\mathbf{N}^2}$$

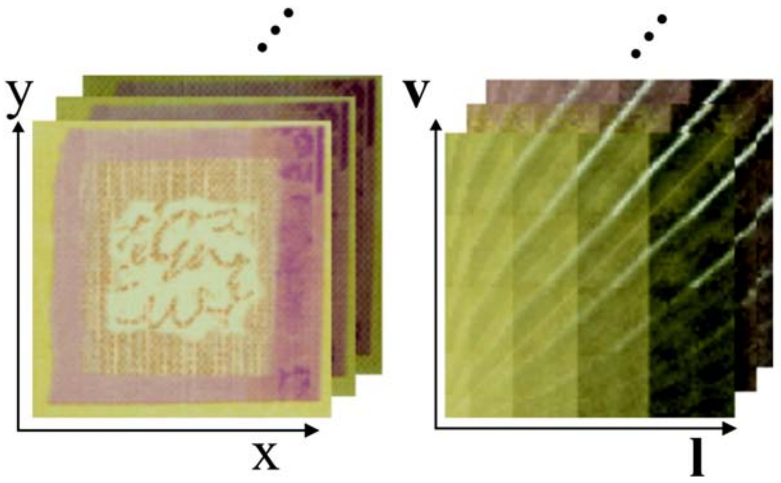


Fig. 2.18: Two arrangements of the BTF data: as set of images (left) and as set of ABRDFs (right).

Note, that ABRDFs do not fulfill physically demanded properties like reciprocity, since they include scattering effects from other parts of the surface.

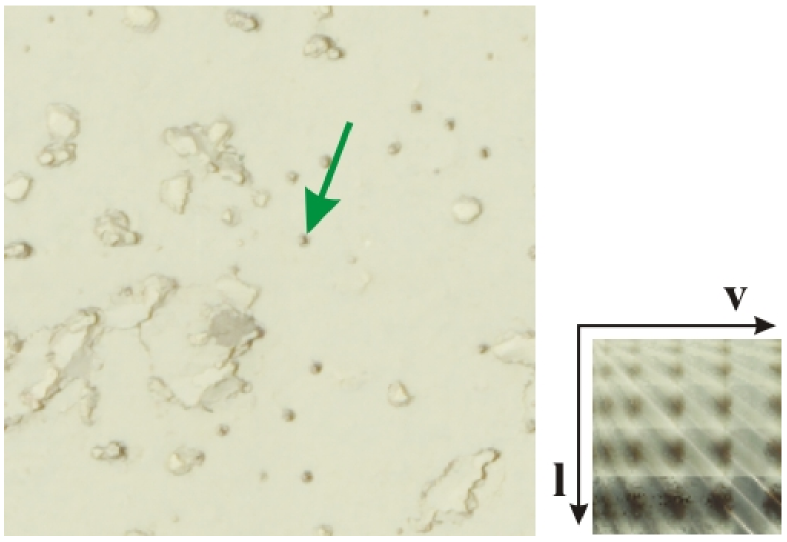


Fig. 2.19: An ABRDF (right) from the PLASTERSTONE BTF (left). While the reflectance of the white plaster alone is quite regular, the holes introduce strong meso-scale shadowing and masking.

As illustrated in Figure 2.19 they can also contain a factor $(\mathbf{n} \cdot \mathbf{l})$ between incident direction and surface normal and strong effects from meso-scale shadowing and masking.

2.1.11 Fitting Analytical BRDF-Models

As mentioned already, BTFs can be understood as spatially varying ABRDFs. Therefore, a natural approach for compressing BTFs is via a pixel-wise representation using BRDF models which are fitted to the synthetic or measured BTF data. Candidate BRDF models need to be efficiently computable to achieve real-time capabilities. Therefore, fitting either the widely used Phong [1975] model, the Blinn [1977] model, the model of Ward [1992] or the Generalized Cosine-Lobe model of Lafortune et al. [1997] to the measured data leads to straightforward extensions from BRDF to BTF representations.

Lafortune Lobes

The simplest BTF model based on analytic function fitting was published by McAllister et al. [2002] and is directly based on the Lafortune model. Lafortune et al. propose to approximate the BRDF by a sum of lobes

$$s(\mathbf{v},\mathbf{l}) = \left(\mathbf{v}^t \cdot \mathbf{M} \cdot \mathbf{l}\right)^n \tag{2.10}$$

with $\mathbf{v}$ and $\mathbf{l}$ denoting local view and light direction respectively, while the general 3×3 matrix $\mathbf{M}$ and the exponent n define the lobe.

To fit these parameters to the reflectance properties of a synthetic or measured BRDF, non-linear fitting methods like the Levenberg-Marquardt algorithm [Press *et al.* 1992] are employed. Fitting the complete matrix allows for very general BRDFs but is very time consuming. Therefore, McAllister et al. decided to employ a more restricted, diagonal matrix $\mathbf{D}$, since fitting and rendering efforts are significantly reduced without major loss in rendering quality. Thus, they use the following, spatially varying lobes:

$$s_{\mathbf{x}}(\mathbf{v},\mathbf{l}) = \left(\mathbf{v}^t \cdot \mathbf{D}_{\mathbf{x}} \cdot \mathbf{l}\right)^{n_{\mathbf{x}}}. \tag{2.11}$$

This results in the following BTF approximation:

$$BTF(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \rho_{d,\mathbf{x}} + \sum_{j=1}^{k} \rho_{s,\mathbf{x},j} \cdot s_{\mathbf{x},j}(\mathbf{v},\mathbf{l}) \tag{2.12}$$

where ρ_d and ρ_s denote diffuse and specular albedo (specified as RGB values) and k is the number of lobes.

The model requires only a few parameters to be stored per pixel resulting in a very compact material representation (about 2 MB per material depending on the spatial resolution and number of lobes). Due to the expensive non-linear minimization, the number of Lafortune lobes is practically limited to about 4 lobes. Therefore the method achieves pleasing results only for a very limited range of materials with minor surface height variation.

Scaled Lafortune Lobes

BRDF models were not designed for the spatially varying ABRDFs which can contain strong effects from meso-scale shadowing, masking (Figure 2.19). Therefore, more specialized models for ABRDFs were developed which also try to model some of these effects.

Daubert et al. [2001] proposed a material representation, which is also based on the Lafortune model but includes an additional, multiplicative term $T_{\mathbf{x}}(\mathbf{v})$ modeling occlusion. Following their proposal, the BTF is evaluated as follows:

$$BTF(\mathbf{x}, \mathbf{v}, \mathbf{l}) \approx T_{\mathbf{x}}(\mathbf{v}) \cdot \left(\rho_{d,\mathbf{x}} + \sum_{j=1}^{k} s_{\mathbf{x},i}(\mathbf{v}, \mathbf{l}) \right). \qquad (2.13)$$

The view-dependent lookup-table T is defined per pixel and therefore the model requires significantly more parameters to be stored. It is thus necessary to combine this method with quantization approaches when handling materials that require significant spatial resolution. The model, as presented originally, was intended to independently represent the three channels of the RGB model by fitting individual Lafortune lobes and lookup-tables for each color channel.

Reflectance Functions

As a qualitative improvement over the previous method, Meseth et al. [Meseth *et al.* 2004b] published an approach to BTF rendering based on fitting a set of functions to the spatially varying reflectance functions of the BTF only and performing a simple linear interpolation for view directions not contained in the measured set. Following this proposal, the BTF is evaluated as follows:

$$BTF(\mathbf{x}, \mathbf{v}, \mathbf{l}) \approx \sum_{v \in N(\mathbf{v})} w_{\mathbf{x},v} \cdot RF_{\mathbf{x},v}(\mathbf{l}) \qquad (2.14)$$

Here, $N(\mathbf{v})$ denotes the set of closest view directions (a subset of the measured view directions), $w_{\mathbf{x},v}$ denotes the spatially varying interpolation weight, and

$RF_{\mathbf{x},v}$ is the spatially varying reflectance function for view direction v which is approximated either by a biquadratic polynomial following the Polynomial Texture Map approach of Malzbender et al. [2001] or adopting Lafortune lobes as follows:

$$RF_{\mathbf{x},v}(\mathbf{l}) \approx \rho_{d,\mathbf{x}} + \rho_{s,\mathbf{x},v}(\mathbf{l}) \cdot \sum_{i=1}^{k} s_{\mathbf{x},v}(\mathbf{l}) \tag{2.15}$$

with $s_{\mathbf{x},v}$ similar to a spatially varying Lafortune lobe but for fixed view direction and k being the number of lobes.

Since the reflectance functions are fitted per pixel and measured view direction, the amount of parameters necessary to evaluate the model is higher than for the scaled Lafortune lobes model. Like the model of McAllister et al. [2002], the approach is designed for efficient rendering and therefore the lobes are intended to compute luminance values that scale the RGB color albedo instead of fitting individual lobes for each color channel. Unlike previous methods based on function fitting, the approach requires interpolation between view directions, since the reflectance functions are defined for fixed view directions.

Reflectance Function Polynomials

Recently, Filip and Haindl [2004] suggested an even more accurate model based on the idea of Lafortune lobes: instead of approximating reflectance functions by summing lobe contributions like Meseth et al. [2004b], they interpolate view and light-dependent polynomials:

$$RF_{\mathbf{x},v}(\mathbf{l}) \approx \sum_{l \in N(\mathbf{l})} w_l \sum_{i=1}^{k} a_{i,\mathbf{x},v,l} \left(\rho_{s,v} \cdot s_{\mathbf{x},v}(\mathbf{l})\right)^{i-1}. \tag{2.16}$$

Here, a denotes the coefficients of the polynomial, $s_{\mathbf{x},v}$ is defined as in equation 2.15 and w_l denotes interpolation weights for the contributions of the nearest light directions $N(\mathbf{l})$ which is a subset of the measured light directions.

Although approximation quality is superior to the previously mentioned approaches based on analytic function fitting and the data requirements are comparable to those of Meseth et al. [2004b], the evaluation of the BTF requires substantially more computation time due to the necessary interpolation of both view and light direction. Especially if applied to each color channel individually, as intended by the authors, this drawback severely limits use in real-time applications. Other applications areas, like texture synthesis - for which the model was intended - or offline rendering, might still find this method useful.

2.1.12 Linear Basis Decomposition

Using parametric BRDF-models fitted to the measured data per pixel has some drawbacks concerning realism. Many models were originally designed to model only a particular class of uniform materials and all models are only an approximation of real reflectance using some simplifying assumptions about the underlying physical process (refer to the recent work of Matusik et al. [Matusik *et al.* 2003] on data-driven reflectance modeling for a more detailed discussion of this topic). The situation becomes even worse for the apparent BRDFs since they contain additional complex effects resulting from the surrounding meso structure.

One way to overcome this problem would be the relaxation of the restricting assumptions of BRDF modeling and the interpretation of the measured data as a multi-dimensional signal. Then general signal-processing techniques such as Principal Component Analysis (PCA) [Press *et al.* 1992] can be applied. PCA minimizes the variance in the residual signal and provides the in a least-squares sense optimal affine-linear approximation of the input signal. The terms PCA and Singular Value Decomposition (SVD) are used synonymously during the rest of this section, since the principal components of the centered data matrix X are the columns of the matrix V with $X = U\Lambda V^T$ being the SVD of X.

PCA has been widely used in the field of image-based rendering to compress the image data. For example Nishino et al. [2001] applied PCA to the reparameterized images of an object viewed from different poses and obtained so-called *eigen-textures*. Matusik et al. [2002] compressed the pixels of the captured reflectance field applying PCA to 8 by 8 image blocks.

The several BTF-compression methods that use PCA differ mainly in two points: (i) the slices of the data to which PCA is applied independently and (ii) how these slices are parameterized.

Per-Texel Matrix Factorization

One approach especially suited for real-time rendering applies PCA to the per-texel ABRDFs. Such methods were developed in the context of real-time rendering of arbitrary BRDFs at the time when the Phong-model was the state of the art in real-time rendering. The original idea as introduced by Kautz and McCool

[1999] can be stated as follows: Given the 4-dimensional BRDF $B_{\mathbf{x}}$, find a factorization into a set of 2-dimensional functions:

$$B_{\mathbf{x}}(\mathbf{v},\mathbf{l}) \approx \sum_{j}^{c} g_{\mathbf{x},j}(\pi_1(\mathbf{v},\mathbf{l}))h_{\mathbf{x},j}(\pi_2(\mathbf{v},\mathbf{l})) \tag{2.17}$$

The functions π_1 and π_2 are projection functions which map the 4D-dimensional BRDF parameters $(\mathbf{v},\mathbf{l})$ to a 2D-space. These projection functions have to be chosen carefully, because the parameterization significantly affects the quality of low-term factorizations. Given such a factorization real-time reconstruction of the BRDF using graphics hardware becomes easy, since the functions $g_{\mathbf{x},j}$ and $h_{\mathbf{x},j}$ can be stored in texture maps and combined during rendering. A trade-off between quality and speed is possible by controlling the number of terms c.

Several methods to find such factorizations have been proposed. Given the sampled values of the BRDF arranged in a 2D-matrix X the SVD of X provides the solution with the lowest RMS-error. But the resulting functions contain negative values which may be problematic for a GPU implementation and the RMS-error may not be the perceptually optimal error metric. As an alternative McCool et al. [2001] presented a technique called Homomorphic Factorization (HF). Instead of using a sum of products they approximate the BRDF by an arbitrary number of positive factors:

$$B_{\mathbf{x}}(\mathbf{v},\mathbf{l}) \approx \prod_{j}^{c} p_{\mathbf{x},j}(\pi_j(\mathbf{v},\mathbf{l})) \tag{2.18}$$

A solution is computed by minimizing RMS-error in the logarithmic space which in fact minimizes *relative* RMS-error in the original minimization problem which was found to be perceptually more desirable. Furthermore, the resulting factors are positive and an arbitrary number of projection functions π_i can be used which allows for highly customized factors that capture certain BRDF features. This is of special importance for the ABRDFs from a measured BTF. They contain horizontal and vertical features like shadowing and masking and also diagonal features like specular peaks. Depending on the parameterization a simple single term expansion can capture only the one or the other.

Recently Suykens et al. [Suykens *et al.* 2003] presented a method called Chained Matrix Factorization (CMF) which encompasses both previous factorization methods by accommodating the following general factorization form:

$$B_{\mathbf{x}}(\mathbf{v},\mathbf{l}) \approx \prod_{j}^{d} \sum_{k}^{c_j} P_{\mathbf{x},j,k}(\pi_{j,1}(\mathbf{v},\mathbf{l})) Q_{\mathbf{x},j,k}(\pi_{j,2}(\mathbf{v},\mathbf{l})). \tag{2.19}$$

Such a chained factorization is computed using a sequence of simple factorizations using for example SVD, but each time in a different parameterization. As a comparison the authors approximated the ABRDFs of synthetic BTFs using CMF and HF. In floating precision they reported similar approximation errors but the factors computed from CMF had a much smaller dynamic range and thus could be safely discretized into 8-bit textures used for rendering on graphics hardware. Furthermore, they stated CMF to be easier to compute and implement.

The compression ratio of per-texel matrix factorization depends on the size of the matrix X, i.e. the sampling of the angular space, and the number of factors. Please note, that these techniques where originally designed for BRDF rendering and for scenes containing a few (maybe some hundred) BRDFs. A single BTF with 256^2 texels contains 64k ABRDFs! Hence to reduce the memory requirements for real-time rendering a clustering of the factors may be necessary.

Full BTF-Matrix Factorization

The per-texel factorization methods from the previous section have the disadvantage, that they do not exploit inter-texel coherence. This can be accomplished by applying a PCA to the complete BTF-data arranged in a $|\mathcal{M}| \cdot |I|$ matrix

$$X = \left(B_{\mathbf{x}_0}, B_{\mathbf{x}_1}, \ldots, B_{\mathbf{x}_{|I|}}\right).$$

Keeping only the first c eigenvalues results in the following BTF-approximation:

$$\mathbf{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \sum_{j}^{c} g_j(\mathbf{x}) h_j(\mathbf{v},\mathbf{l}) \tag{2.20}$$

This approach was used in the works of Liu et al. [2004] and Koudelka et al. [2003].

The remaining issue is, how to choose c. Ravi Ramamoorthi [2002] showed by an analytic PCA construction, that using about five components is sufficient to reconstruct lighting variability in images of a convex object with Lambertian reflectance. Therefore, for nearly diffuse and relatively flat samples a good reconstruction quality can be expected for low c. However, as illustrated in Figure 2.20, this will

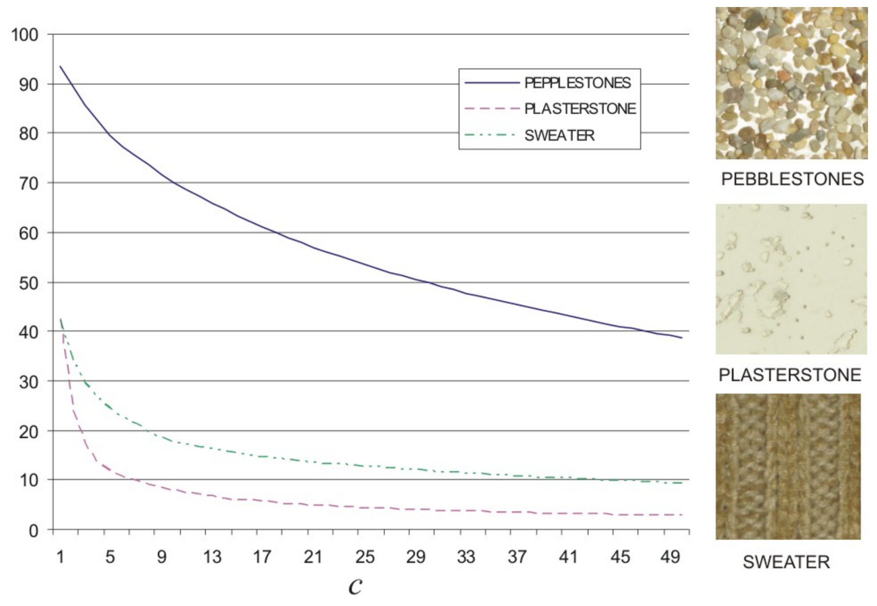


Fig. 2.20: RMS-error of the full BTF-matrix factorization depending on the number of terms c. There is a direct correspondence between the magnitude and decrease of error and the BTF-complexity.

not be sufficient for complex BTFs containing considerable self-shadowing, masking and obviously for non-diffuse reflectance.
Therefore, Koudelka et al. chose $c = 150$ to represent all significant effects with enough fidelity. To reduce the size of the resulting dataset even further they stored the basis-vectors as JPG-images resulting in very high compression rates. But of course real-time reconstruction from this representation is not possible.

An alternative approach for full BTF-matrix factorization was presented by Vasilescu and Terzopoulos [Vasilescu & Terzopoulos 2004]. They arranged the BTF-data in a 3-mode tensor and applied multi-linear analysis (3-mode SVD), which corresponds to the application of standard SVD to different arrangements of the data. It is worth noting, that the resulting reconstruction formula provides a more flexible dimensionality reduction by allowing to reduce the represented variation in view and lighting directions independently. Compared to standard PCA with the same number of components the methods leads to a higher RMS-error, but the authors claim that keeping more variation in the viewing direction gives perceptually more pleasing results.

A serious problem of the full BTF-matrix factorization methods is the size of the matrix X which easily could reach several gigabytes in float-precision. In this case, even the computation of the covariance matrix XX^T would be a very lengthy operation and special out-of-core routines have to be implemented. As an alternative the factorization can be computed for a subset of the data only.

Per-View Factorization

The full BTF-matrix factorization suffers from memory problems during computation and reconstruction is only fast *and* correct for relatively simple materials. Therefore, Sattler et al. [Hauth *et al.* 2002; Sattler *et al.* 2003] published a method that deals with these problems. Because their original intention was to visualize cloth BTFs that exhibit a significant amount of depth variation and hence highly non-linear view-dependence, they use an approach similar to the work of Meseth et al. [Meseth *et al.* 2004b] in the sense that they applied PCA to slices of the BTF with fixed view-direction. This leads to the following set of data-matrices:

$$X_{\mathbf{v}_i} := \left(T_{(\mathbf{v}_i,\mathbf{l}_0)}, T_{(\mathbf{v}_i,\mathbf{l}_1)}, \ldots, T_{(\mathbf{v}_i,\mathbf{l}_{M_{\mathbf{v}_i}})}\right)$$

with $M_{\mathbf{v}_i}$ denoting the number of sampled light directions for the given view direction $\mathbf{v}_i$. The PCA is applied to all matrices $X_{\mathbf{v}_i}$ independently which poses no computational problems compared to the full BTF matrix. Then keeping only the first c eigenvalues gives the following BTF approximation:

$$\mathbf{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \sum_{j=1}^{c} g_{\mathbf{v},j}(\mathbf{l}) h_{\mathbf{v},j}(\mathbf{x}) \tag{2.21}$$

Compared to equation 2.20, the value of c can be set much lower (the authors chose c between 4 and 16) which enables interactive or even real-time rendering frame rates with good visual quality. However, the memory requirements are much higher. For example, $c = 16$ and $M_{\mathbf{v}_i} = 81$ lead to more than 1200 terms that have to be stored.

Per-Cluster Factorization

As already mentioned, a complex BTF contains highly non-linear effects like self-shadowing, self-occlusion and non-diffuse reflectance. Nevertheless, many high-dimensional data sets exhibit a local linear behavior. Applying per-texel or per-view factorization implicitly exploits this observation by selecting fixed subsets of the data and approximating these subsets with an affine-linear subspace. A more general approach would choose these subsets depending on the data. This is the idea behind the local PCA method, which was introduced by Kambhatla and Leen [Kambhatla, N. & Leen, T.K. 1997] to the machine-learning community in competition to classical non-linear/neural-network learning algorithms. It combines clustering and PCA using the reconstruction error as metric for choosing the best cluster.

Recently, Müller et al. [Müller *et al.* 2003] applied this method to BTF-compression and proposed the following approximation:

$$\mathbf{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \sum_{j}^{c} g_{k(\mathbf{x}),j}(\mathbf{x}) h_{k(\mathbf{x}),j}(\mathbf{v},\mathbf{l}) \tag{2.22}$$

The operator $k(\mathbf{x})$ is the cluster index look-up given a position $\mathbf{x}$. In this case clustering is performed in the space of ABRDFs which was found being better suited for real-time rendering than clustering in the space of images. Now the number of clusters can be chosen according to computational resources and quality requirements. Figure 2.21 compares per-cluster factorization to full matrix factorization with the same number of terms c. Good results were obtained for cluster counts between 16 and 32, which is much smaller than the fixed cluster number (e.g. $M_{\mathbf{v}_i} = 81$) used in per-view factorization.

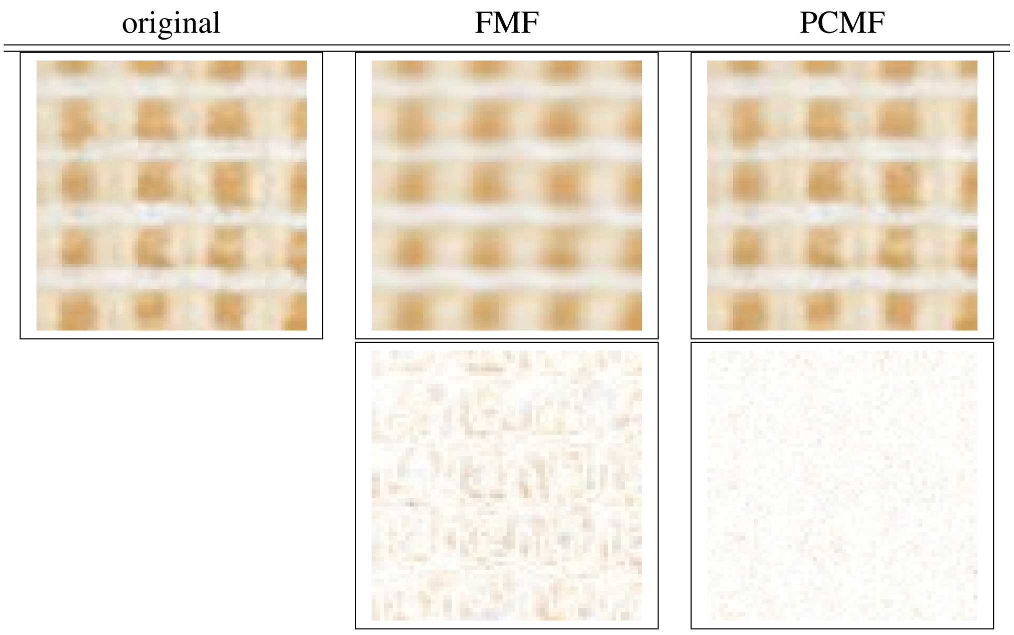


Fig. 2.21: Comparing full matrix factorization and per-cluster matrix factorization. From left to right: original frontal view of PROPOSTE BTF, reconstruction from full matrix factorization (FMF) with $c = 8$ terms, reconstruction from per-cluster matrix factorization (PCMF) with 32 clusters and 8 terms per cluster. Second row: enhanced and inverted difference images.

Discussion

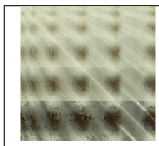
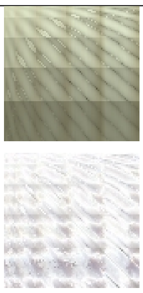
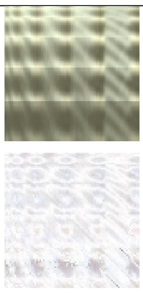
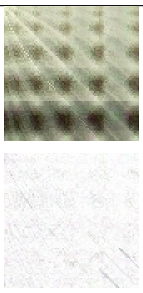
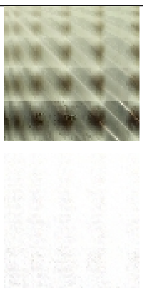
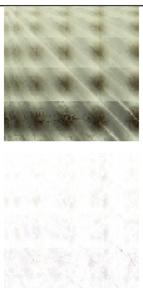
original	SLAF	RF	CMF	PVMF	PCMF
					
1.2 GB	32 MB	106 MB	60 MB	121 MB	11 MB

Fig. 2.22: Comparison of selected BTF-compression methods. Top row: original and reconstructed ABRDFs. Second row: inverted difference images. Bottom row: storage requirements for the compressed representations using parameters suitable for interactive rendering. From left to right: Original ABRDF of PLASTERSTONE BTF, Scaled Lafortune Lobes with 2 lobes (SLAF), Reflectance Functions with per-view polynomial (RF), Per-Texel Chained Matrix Factorization with 4 factors (CMF), Per-View Matrix Factorization with 8 terms (PVMF), Per-Cluster Matrix Factorization with 32 clusters and 8 terms (PCMF). The original dataset consists of 6561 8-bit RGB-images with 256^2 pixels in size. Numerical precision is 16 bit for PCMF and 8 bit for all other methods.

Figure 2.22 shows a comparison of several methods discussed above. Obviously, the quality of the reconstruction from linear basis decompositions is better than from parametric reflectance models even if additional parameters like scaling values per view or even a full fit per view are used. Furthermore, the increased quality achieved by using this additional complexity does not legitimate the increased memory requirements. The qualitatively best result is achieved using per-view factorization but unfortunately the memory requirements are very high, since for every measured view direction a set of textures and weights has to be stored. Using per-texel matrix factorization does not exploit spatial coherence and thus the quality is not as good as per-view or per-cluster factorization even for considerable memory requirements. Furthermore, the chained resampling steps can introduce additional resampling error. Suykens et al. [Suykens *et al.* 2003] propose k-means clustering of the factors across the spatial dimension to reduce memory requirements. This obviously could be applied to every per-texel BTF compression method, but for complex materials that do not contain uniform areas this will introduce cluster artifacts. These cluster artifacts are reduced using PCA in each cluster as done in the per-cluster factorization method. Hence, this method seems to offer a good compromise between reconstruction cost, visual quality and memory requirements.

In conclusion, current acquisition systems are expensive and the measurement process is time consuming as the directional dependent parameters (light- and view-direction) have to be controlled very accurately. Otherwise the resulting data will be incorrect. Furthermore, the size of measured BTFs lies in a range from hundreds of megabytes to several gigabytes. This hampers both synthesis and rendering so that only effective compression techniques can provide a solution.

Due to these limitations BTF rendering is still not mature enough for industrial applications. Nevertheless, there is a growing demand for interactive photo-realistic material visualization in the industry. For special applications such as high-end virtual reality environments, BTF rendering can already satisfy these demands. Simple material representations like 2-D texture or bump-maps sooner or later will be replaced by more complex representations that reproduce all the subtle effects of general light-material interaction.

The acquisition of the BTF of real world materials requires a complex and controlled measurement environment. As BTF acquisition is physical measurement of real-world reflection, special attention has to be paid to the device calibration and image registration. Otherwise the measurements will contain inaccuracies which may generate visible rendering artifacts.

2.1.13 Rendering

Generally accurate and physically plausible rendering algorithms have to compute a solution of the rendering equation at every surface point $\mathbf{x}$ (neglecting emissive effects):

$$L_r(\mathbf{x},\mathbf{v}) = \int_{\Omega_i} \rho_{\mathbf{x}}(\mathbf{v},\mathbf{l}) L_i(\mathbf{x},\mathbf{l})(n_{\mathbf{x}} \cdot \mathbf{l})\, d\mathbf{l} \tag{2.23}$$

Here, $\rho_{\mathbf{x}}$ denotes the BRDF, L_i denotes incoming radiance, $n_{\mathbf{x}}$ is the surface normal and Ω_i is the hemisphere over $\mathbf{x}$.
Including measured BTFs into the rendering equation 2.23 is very simple:

$$L_r(\mathbf{x},\mathbf{v}) = \int_{\Omega_i} BTF(\mathbf{x},\mathbf{v},\mathbf{l}) L_i(\mathbf{x},\mathbf{l})(n_{\mathbf{x}} \cdot \mathbf{l})\, d\mathbf{l} \tag{2.24}$$

Now the measured BRDF at point $\mathbf{x}$ is simply looked up from the BTF. It is assumed, that a mapping from the 3D-surface to the 2D spatial texture domain already exists. Please note that the BTF also models meso-scale geometry. However, since this information is projected into the BTF the rendering will not be correct for example at object silhouettes.

2.1.14 Solving the Rendering Equation

Currently there are two popular approaches that are primarily used to solve the rendering equation.

Monte-Carlo Sampling

The first approach tries to solve equation 2.23 accurately using Monte-Carlo sampling methods. Many variants of this approach such as path tracing [Kajiya 1986], distribution ray tracing [Cook *et al.* 1984] and photon mapping [Jensen, H.W. 1996], to mention a few, have been developed over the years. Despite recent advances towards interactive global illumination (for example [Wald *et al.* 2003b]) accurate solutions of arbitrary complex scenes can still take hours or even days to compute.

Obviously equation 2.24 can be solved using Monte-Carlo sampling as well. In this case the renderer simply has to be extended to support a particular BTF compression scheme which in fact corresponds to the implementation of the decompression stage on the CPU. This is possible for any compression method introduced in Section 2.1.10.

Approximate Solutions for Real-Time Rendering

The second approach makes a priori several simplifying assumptions so that the integral can be solved more quickly and is amenable to hardware implementation. The goal is to reduce the integral in equation 2.23 to a finite sum containing only very few terms.

Point Lights The most popular simplification is using only a set $\Lambda = \{\mathbf{l}_j\}$ of point or directional light-sources and discarding general interreflections (i.e. the recursion in equation 2.23). For notational simplicity the term $\mathbf{l}_j$ encodes both intensity and direction of the light source. Since these lights are given in global coordinates and the BRDF is usually given in the local frame at $\mathbf{x}$, the local coordinates $\tilde{\mathbf{l}}$ are also needed:

$$L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{j}^{|\Lambda|} \rho_{\mathbf{x}}(\mathbf{v}, \tilde{\mathbf{l}}_j) G(\mathbf{x}, \mathbf{l}_j)(n_{\mathbf{x}} \cdot \mathbf{l}_j)\mathbf{l}_j \tag{2.25}$$

The geometry term $G(\mathbf{x}, \mathbf{l}_j)$ contains the visibility function and an attenuation factor depending on the distance. In the following, the visibility function in the geometry term is neglected, because interactive large-scale shadowing is an independent research topic in its own right (for a survey refer for example to [Hasenfratz *et al.* 2003]). Using equation 2.25 a scene can be rendered real-time using today´s consumer graphics hardware. Arbitrary BRDFs can be implemented using the programmable vertex- and fragment-shader capabilities. In the case of BTF-rendering the following sum has to be computed:

$$L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{j}^{|\Lambda|} BTF(\mathbf{x}, \mathbf{v}, \tilde{\mathbf{l}}_j) G(\mathbf{x}, \mathbf{l}_j)(n_{\mathbf{x}} \cdot \mathbf{l}_j)\mathbf{l}_j \tag{2.26}$$

In fact, the challenge of developing a fast BTF-rendering algorithm for point lights is reduced to the implementation of a fragment program that evaluates the BTF for given parameters. For several of the compression schemes presented in Section 2.1.10 such an implementation has been proposed. Details can be found in Section 2.1.17.

Infinitely Distant Illumination Another widely used simplification assumes infinitely distant incident illumination and no interreflections. In this case the dependency of incident lighting on surface position $\mathbf{x}$ can be removed:

$$L_r(\mathbf{x}, \mathbf{v}) \approx \int_{\Omega_i} \rho_{\mathbf{x}}(\mathbf{v}, \mathbf{l}) L_i(\mathbf{l})(n_{\mathbf{x}} \cdot \mathbf{l})\, d\mathbf{l} \tag{2.27}$$

For special types of BRDFs this integral can be precomputed (for example [Kautz *et al.* 2005]) and the results can be displayed in real-time. Another approach is to reduce the integral to a finite sum by projecting light and BRDF onto a basis like Spherical Harmonics (SH) [Sloan *et al.* 2002] or wavelets [Ng *et al.* 2003] and keeping only a finite number of terms. Using this approach even transport effects like shadows and interreflections can be precomputed and projected onto the basis.

Special care has to be taken while transferring such an approach to BTF-rendering. The methods were originally designed only for special types of BRDFs or the results are only computed per vertex. Hence, only few algorithms for BTF-rendering using distant illumination have been published so far. The details will be discussed in Section 2.1.18.

2.1.15 BTF-Rendering using Real-Time Raytracing

The recent advances in computation power and improved algorithms allow for interactive ray-tracing even on a single desktop PC [Wald *et al.* 2003b]. Real-time performance can be achieved using PC clusters. As in Section 2.1.14 the integration of a particular BTF compression scheme into such a system corresponds to the implementation of the decompression stage on the CPU.

2.1.16 BTF-Rendering using Graphics Hardware

To incorporate BTFs into current real-time rendering systems, the evaluation of the BTF should be done on the graphics processing unit (GPU) i.e. integrated into the fragment-shading process. To achieve this, the compressed representation of the BTF must be stored in textures and the reconstruction is performed using the programmable units of the graphics board. The parameters for BTF evaluation are the standard 2D-texture coordinates $\mathbf{x}$, the local view direction $\mathbf{v}$ and in the case of point light sources the local light direction $\mathbf{l}$.

A crucial point in all BTF-rendering methods is interpolation. Since a measured BTF is discrete in all 6 dimensions, smooth renderings require interpolation in each dimension. To achieve high frame-rates it is indispensable, that at least some of these dimensions are interpolated using built-in hardware interpolation. For the other dimensions either the nearest neighbor must be chosen or the interpolation of the nearest neighbors has to be executed explicitly in the fragment shader. In both cases there has to be an operator $N(\cdot)$ that supplies the nearest samplings. Such a look up can be performed on the GPU using dependent texture look-ups. To perform explicit interpolation in the fragment shader, the corresponding interpolation weights τ_i should also be precomputed and stored in textures.

In the following sections existing BTF-rendering methods will be presented, that achieve interactive or even real-time frame-rates exploiting the capabilities of current graphics hardware.

2.1.17 Interactive Rendering of BTFs with Point Lights

Parametric Reflectance Models

Efficient implementations of the parametric reflectance models from Section 2.1.11 have been presented by various publications. McAllister et al. [2002] describes a real-time evaluation scheme for equation 2.12. Coefficients of the spatially varying Lafortune lobes are stored in 2D textures. Evaluation can efficiently be done using vertex and fragment shaders. Since Lafortune lobes are continuous in the angular domain, no interpolation is required in the angular domain. Spatial interpolation has to be done explicitly in the fragment shader (magnification) or is left to the multisampling and built-in MIP-mapping capabilities of graphics hardware, although MIP-maps have to be built manually (for example by fitting new sets of Lafortune lobes for each BTF resolution).

As an extension to the model of McAllister et al., the BTF model of Daubert et al. [2001] only requires an additional evaluation of the view-dependent visibility term. Although significant numbers of parameters are required to store this term, it can easily be encoded in a texture. Interpolation of the view-direction can be achieved using graphics hardware. Spatial interpolation is done like in the previous approach.

The even more complex model of Meseth et al. [2004b] is evaluated very similarly to the previous two ones with the significant difference that the discretization of the view direction requires an additional manual interpolation (as denoted in equation 2.14). Therefore, two cube maps are utilized which store for each texel in the cube map (representing a certain view direction) the three closest view directions and respective interpolation weights. Interpolation is achieved by evaluating the reflectance functions for the three closest view directions and interpolating the results based on the interpolation factors stored in the cube map. Spatial interpolation can be done exactly like in the previous approaches.

Efficiently evaluating the BTF model of Filip and Haindl [2004] constitutes an even more complex problem since it requires interpolation of both view and light direction, effectively requiring evaluation of the basic model nine times. Since the model was not intended for real-time rendering, no such algorithm was proposed yet and it is questionable if the improved rendering quality can compensate the significantly increased rendering costs.

Per-Texel Matrix Factorization

Generally, the rendering algorithms for BRDF-factorizations can be used [Kautz & McCool 1999; McCool *et al.* 2001] with the difference, that the factors now change in every fragment. Suykens et al. [2003] detailed how this can be accomplished:
Every factor is reparameterized and stored into a parabolic map [Heidrich & Sei-

del 1998]. Then all these factors are stacked into 3D-textures and normalized to a range between 0 and 1. The resulting scale factors are stored in a scale map. A particular factor is selected in its 3D-texture using transformed 2D texture coordinates or, if clustered factors are employed, by the values from an index map. To avoid mixing of neighboring factors in the 3D-texture the z-coordinate has to be chosen carefully. A value within a particular factor is indexed using the appropriately reparameterized local view and light directions. While interpolation inside a factor is supported by the hardware, spatial interpolation between neighboring texels is not implemented. equation 2.19 now can be executed in a fragment shader.

Per-View Matrix Factorization

Sattler et al. [2003] demonstrated how equation 2.21 can be implemented using a combination of CPU and GPU computations. Figure 2.23 shows the basic approach: Combination of the eigen-textures $h_{\mathbf{v},j}(\mathbf{x})$ with the appropriate weights $g_{\mathbf{v},j}(\mathbf{l})$ using the multi-texturing capabilities of modern graphics boards. This combination is done for every triangle-vertex. A smooth transition is ensured by blending the resulting three textures over the triangle using a fragment program [Chen *et al.* 2002].

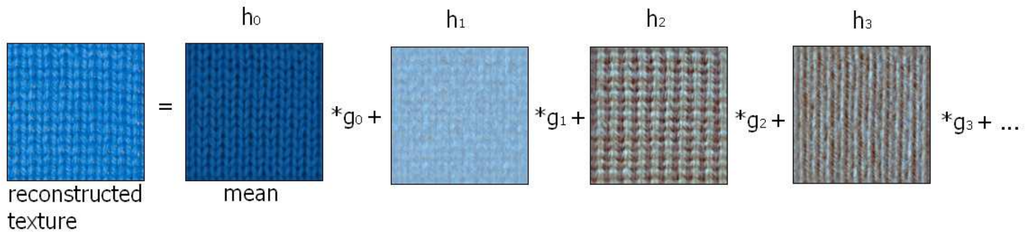


Fig. 2.23: Schematic rendering using the per-view matrix factorization.

While the interpolation in the spatial domain is done by the graphics hardware, light and view direction are interpolated explicitly. To interpolate between the nearest measured light directions the term

$$\mathbf{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \sum_{\tilde{l}\in N(\mathbf{l})} \tau_{\tilde{l}} \sum_{j=1}^{c} g_{\mathbf{v},j}(\tilde{l}) h_{\mathbf{v},j}(\mathbf{x}) \tag{2.28}$$

has to be evaluated. In order to speed up this computation the weights $\lambda_{\mathbf{v},j}(\mathbf{l}) = \sum_{\tilde{l}\in N(\mathbf{l})} \tau_{\tilde{l}} \cdot g_{\mathbf{v},j}(\tilde{l})$ are computed on the CPU per frame and sent to the GPU resulting in the term

$$\mathbf{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \sum_{j=1}^{c} \lambda_{\mathbf{v},j}(\mathbf{l}) h_{\mathbf{v},j}(\mathbf{x}). \tag{2.29}$$

To perform view-interpolation different sets of eigen-textures have to be combined resulting in an expensive combination:

$$\mathbf{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) \approx \sum_{\tilde{v}\in N(\mathbf{v})} \tau_{\tilde{v}} \sum_{j=1}^{c} \lambda_{\tilde{v},j}(\mathbf{l}) h_{\tilde{v},j}(\mathbf{x}). \tag{2.30}$$

Full BTF-Matrix Factorization

To evaluate equation 2.20 the factors $g_j(\mathbf{x})$ and $h_j(\mathbf{v},\mathbf{l})$ have to be evaluated, whereas the factors $g_j(\mathbf{x})$ can be stored as simple 2D-textures and the factors $h_j(\mathbf{v},\mathbf{l})$ as 4D-textures. But unfortunately neither 4D-textures nor their full interpolation is currently supported by graphics hardware.

Therefore, Liu et al. [2004] store the factors $h_j(\mathbf{v},\mathbf{l})$ into stacks of 3D-textures. The tri-linear filtering capabilities of graphics hardware now can be exploited to interpolate the view direction and the polar angle of the light direction. The final step for 4D filtering is performed manually by blending two, tri-linearly filtered values with closest azimuth angles in light direction. As usually, the values of $h_j(\mathbf{v},\mathbf{l})$ parameterized over the hemispheres of view and light directions have to be resampled and reparameterized in order to be stored in textures. In order to avoid the fragment shader's online effort of calculating the reparameterized local light and view directions, which are necessary for accessing the 3D-textures and interpolation weights, the mapping is precomputed and stored in a cube map.

Per-Cluster Matrix Factorization

Apart from the additional cluster look-up, evaluating equation 2.22 is essentially the same as evaluating 2.20. Hence the real-time rendering algorithm for equation 2.22 presented by Schneider [2004] is similar in style to the approach presented in the previous section. He also stores the factors $h_{k(\mathbf{x}),j}(\mathbf{v},\mathbf{l})$ in stacks of 3D-textures and accesses them through reparameterized local light and view directions. The cluster index introduces an additional dependent texture look-up. Since existing graphics boards only support hardware supported interpolation of fixed point values, every factor is quantized to 8-bit separately yielding scaling factors $s_{j,k}$. As compensation the factors $g_j(\mathbf{x})$, which can be stored as floating-point values since they require no interpolation, have to be divided by the corresponding scaling factor $s_{j,k}$.

Mipmapping the BTF can simply be implemented by executing the shader instructions twice (once for the currently best mipmap level and once for the next best mipmap level) and interpolating the resulting colors. Bilinear, spatial interpolation is currently not supported, since the additional overhead is prohibitive. Fortunately, hardware supported full-screen antialiasing can reduce potential artifacts significantly.

2.1.18 Interactive Rendering of BTFs with Distant Illumination

Relighting of BTFs with distant illumination is considered according to equation 2.27. Typically this distant illumination is represented by an environment map. Such an environment map can either be computed by the graphics hardware or captured from a natural environment by taking pictures (for example of a metallic sphere) [Debevec & Malik 1997; Lightprobes 2005].

Parametric Reflectance Models

Combining parametric reflectance models for BTFs with image-based lighting relies on the concept of prefiltered environment maps, which was first applied to the diffuse reflection model by Miller and Hoffman [1984] and Greene [1986]. For a diffuse BTF with spatially varying reflection coefficients ($\mathbf{BTF}(\mathbf{x},\mathbf{v},\mathbf{l}) = \rho_d(\mathbf{x})$), equation 2.27 reduces to:

$$\begin{aligned} L_r(\mathbf{x},\mathbf{v}) &= \int\limits_{\Omega_i} \rho_d(\mathbf{x}) L_i(\mathbf{l})(\mathbf{n}_x \cdot \mathbf{l})\, d\mathbf{l} \\ &= \rho_d(\mathbf{x}) \int\limits_{\Omega_i} L_i(\mathbf{l})(\mathbf{n}_x \cdot \mathbf{l})\, d\mathbf{l} \\ &= \rho_d(\mathbf{x}) D(\mathbf{n}_x). \end{aligned} \tag{2.31}$$

The prefiltered environment map $D(\mathbf{x},\mathbf{n}_x)$ can be precomputed on the CPU, stored in a cube texture map and used during rendering. Kautz and McCool [Kautz & McCool 2000] extended the concept to isotropic BRDFs. Since for non-diffuse cases the prefiltered result becomes view-dependent, they approximated equation 2.27 as follows:

$$
\begin{aligned}
L_r(\mathbf{x},\mathbf{v}) &\approx (\mathbf{n}_x \cdot p(\mathbf{x},\mathbf{v})) \int\limits_{\Omega_i} BTF(\mathbf{x},\mathbf{v},\mathbf{l}) L_i(\mathbf{l})\, d\mathbf{l} &\quad (2.32)\\
&\approx (\mathbf{n}_x \cdot p(\mathbf{x},\mathbf{v}))\, S(\mathbf{x},\mathbf{v}) &\quad (2.33)
\end{aligned}
$$

where $p(\mathbf{x},\mathbf{v})$ denotes the peak direction, i.e. the light direction with maximum influence on $L_r(\mathbf{x},\mathbf{v})$. The accuracy of this approximation increases with the specularity of the spatially varying ABRDFs. With this assumption, the specular prefiltered environment $S(\mathbf{x},\mathbf{v})$ can be computed analogously to the diffuse case.

McAllister et al. [2002] applied this concept to equation 2.12. The diffuse and specular terms are considered individually resulting in two prefiltered environment maps: the diffuse $D_{\mathbf{x}}(\mathbf{n}_x)$ and a specular one, which is computed based on the ideas of Kautz and McCool [Kautz & McCool 2000]. Note that whenever $\mathbf{x}$ is written as index, it refers to instances discretely sampled in the spatial domain. First, the peak direction

$$p_{\mathbf{x}}(\mathbf{v}) = \mathbf{v}^t \cdot \mathbf{D}_{\mathbf{x}}$$

of each lobe is computed. Then the specular illumination part (for ease of notation, a 1-lobe approximation is assumed) is rewritten as follows:

$$
\begin{aligned}
L_{r,s}(\mathbf{x},\mathbf{v}) &\approx \int\limits_{\Omega_i} \rho_{s,\mathbf{x}} \cdot (p_{\mathbf{x}}(\mathbf{v}) \cdot \mathbf{l})^{n_{\mathbf{x}}} L_i(\mathbf{l})(\mathbf{n}_x \cdot \mathbf{l})\, d\mathbf{l}\\
&\approx \rho_{s,\mathbf{x}} \cdot (\mathbf{n}_x \cdot p_{\mathbf{x}}(\mathbf{v})) \int\limits_{\Omega_i} (p_{\mathbf{x}}(\mathbf{v}) \cdot \mathbf{l})^{n_{\mathbf{x}}} L_i(\mathbf{l})\, d\mathbf{l}\\
&\approx \rho_{s,\mathbf{x}} \cdot (\mathbf{n}_x \cdot p_{\mathbf{x}}(\mathbf{v}))\, ||p_{\mathbf{x}}(\mathbf{v})||^{n_{\mathbf{x}}} S\left(p_{\mathbf{x}}(\mathbf{v}), n_{\mathbf{x}}\right) &\quad (2.34)
\end{aligned}
$$

with

$$S(p_{\mathbf{x}}(\mathbf{v}), n_{\mathbf{x}}) = \int\limits_{\Omega_i} \left(\frac{p_{\mathbf{x}}(\mathbf{v})}{||p_{\mathbf{x}}(\mathbf{v})||} \cdot \mathbf{l}\right)^{n_{\mathbf{x}}} L_i(\mathbf{l})\, d\mathbf{l} \quad (2.35)$$

$S(\mathbf{p}, n)$ denotes the specular prefiltered environment map.

Evaluating Approximation 2.34 can efficiently be done using fragment shaders by first computing $p_{\mathbf{x}}(\mathbf{v})$, looking up the respective specular prefiltered value from a cube map, evaluating the specular part and adding the diffuse contribution. Special care has to be taken only concerning the specular exponent, which is represented as discrete versions in $S(\mathbf{p}, n)$ only. One can either choose the closest exponent or interpolate from the two closest ones.

Like for point-like sources, the continuity of the Lafortune lobes in the angular domain requires additional interpolation in the spatial domain only. Again, graphics hardware features like multisampling and MIP-mapping are employed for this task.

Whereas the extension of this approach to the model of Daubert et al. [2001] requires an additional evaluation of the visibility term only, extending it to the reflectance function based model of Meseth et al. [Meseth *et al.* 2004b] requires interpolation from the results of the reflectance functions corresponding to the closest measured view directions. Nevertheless, all three approaches allow for real-time rendering.

Bi-Scale Radiance Transfer

Precomputed Radiance Transfer (PRT), as originally introduced by Sloan et al. [2002], is evaluated per vertex only and interpolated across the triangle. In a follow up work, Sloan et al. [2003a] extended PRT to support also spatially varying reflectance across the triangle. They projected the BTF per pixel and fixed view direction onto the SH basis generating a Radiance Transfer Texture (RTT) which now represents the per-view response of the material to the spherical harmonics basis. To cover large geometry with the memory-intensive texture they used the synthesis algorithm of Tong et al. [2002] to generate an ID-map over the mesh which references into the RTT.

Generation of the RTT and the ID-map The generation of the RTT $B(\mathbf{x}, \mathbf{v})$ is accomplished by projecting the BTF onto the first c elements of the SH basis $\{Y_j\}_{j\in\mathbf{N}}$:

$$B(\mathbf{x}, \mathbf{v})_j = \int_{\Omega_i} BTF(\mathbf{x}, \mathbf{v}, \mathbf{l}) Y_j(\mathbf{l}) \, d\mathbf{l} \tag{2.36}$$

Thus, the RTT is a 4D-array of c Spherical Harmonics coefficients (typically $c = 25$).

The ID-map is generated using BTF synthesis over densely resampled geometry called a *meso-mesh*. This step assigns every vertex an ID into the RTT. Then a texture atlas for the original coarse mesh is generated and for every texel in this atlas the nearest vertex in the meso-mesh is found and the corresponding ID is assigned to the texel.

Real-Time Rendering Standard PRT-rendering is performed per-vertex. That means computation of a matrix-vector multiplication between the transfer matrix and the incident lighting SH-coefficients. The resulting transferred lighting vector is interpolated across the triangle. To compute exitant radiance per-fragment the interpolated transferred lighting vector is dot multiplicated with the corresponding vector in the RTT which is accessed by the ID map and the local view direction. This is done in a fragment program. Each texel of the RTT is stored in an 8x8 texture block encoding the view dependence. This allows smooth bilinear interpolation across views using built-in hardware interpolation. Interpolation between spatial samples is not performed.

Since the RTTs are not compressed, the method supports only sparse samplings. They used 8x8 view-samples and 64x64 spatial samples which results in $64^3 * 25$ SH-coefficients that have to be stored per color band.

Per-View Matrix Factorization

Sattler et al. [2003] also proposed a method to relight the per-view factorized BTFs could by environment maps. The main idea is to discretize the integral in equation 2.27 using a hemicube [Cohen & Greenberg 1985], which leads to

$$L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{\hat{l} \in H_i} \rho_{\mathbf{x}}(\mathbf{v}, \hat{l}) H_{i,\mathbf{x}}(\hat{l})(n_{\mathbf{x}} \cdot \hat{l}) \tag{2.37}$$

where $H_{i,\mathbf{x}}$ denotes the discretized hemicube. $H_{i,\mathbf{x}}(\hat{l})$ returns the color in the hemicube over $\mathbf{x}$ at direction $\hat{l}$ or zero if the direction is occluded. The hemicube is precomputed and stored in a visibility map as follows:

Hemicube Precomputation The hemicube $H_{i,\mathbf{x}}$ stores a discretization of the hemisphere at the vertex $\mathbf{x}$. Figure 2.24 (left) shows an unfolded hemicube. Using a color-coded environment map (Figure 2.24 middle) a look-up table into a high dynamic range map (Figure 2.24 right) is created. This allows easy exchange of the environment map. By rendering the geometry in white color into the hemicube the visibility function is computed and self-shadowing can be supported. Since the directions in the visibility map not necessarily correspond to the measured light directions in the BTF, the map furthermore stores the four nearest measured light directions and the corresponding interpolation weights.
A hemicube pixel now stores the following information:

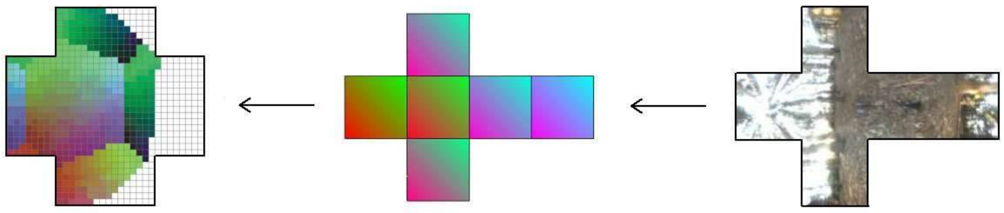


Fig. 2.24: Hemicube computation. Visibility map (left) with rendered color-coded lookup environment map (middle). White color in the visibility map stands for occlusion caused by the mesh. On the right side a HDR environment is shown, which is mapped onto the color-coded one.

- visibility of a pixel of the environment map and if it is visible, the position of this pixel in the map
- four nearest measured directions with respect to the direction represented by this pixel
- corresponding interpolation weights

Rendering algorithm Given the nearest measured view direction $\mathbf{v}$ at vertex $\mathbf{x}$ and substituting the BRDF in equation 2.37 by the per-view factored BTF representation including the foreshortening term yields the following sum:

$$L_r(\mathbf{x},\mathbf{v}) \approx \sum_{\mathbf{l}\in H_i} \left(\sum_{j=1}^{c} \lambda_{\mathbf{v},j}(\mathbf{l}) h_{\mathbf{v},j}(\mathbf{x}) \right) H_{i,\mathbf{x}}(\mathbf{l}) \tag{2.38}$$

The factors $\lambda_{\mathbf{v},j}(\mathbf{l})$ are from equation 2.29. As in equation 2.28 the factors $\gamma_{\mathbf{v},j} = \sum_{\mathbf{l}\in H_i} \lambda_{\mathbf{v},j}(\mathbf{l}) H_{i,\mathbf{x}}(\mathbf{l})$ are precomputed per vertex and sent to the GPU where the final expression is evaluated:

$$L_r(\mathbf{x},\mathbf{v}) \approx \sum_{j=1}^{c} \gamma_{\mathbf{v},j} h_{\mathbf{v},j}(\mathbf{x}) \tag{2.39}$$

which again is only a linear combination of basis textures. For view interpolation the same calculations as in equation 2.30 have to be applied.
Since $\gamma_{\mathbf{v},j}$ is computed for all vertices $\mathbf{x}$ of the geometry a vector U holding all $\gamma_{\mathbf{x},\mathbf{v},\mathbf{j}}$ can be introduced:

$$U = (\gamma_{\mathbf{x}_0,\mathbf{v}_0,1}, \ldots, \gamma_{\mathbf{x}_0,\mathbf{v}_0,c}, \ldots) \tag{2.40}$$

This vector has do be calculated once per environment map and allows real-time changes of the viewing position. A drawback of this method is, that changing the environment map and even a simple rotation implies a complete recalculation of U. Depending on the visibility map resolution and the number of vertices, this computation can take too long for interactive change of lighting. Reducing the visibility map resolution adaptively to achieve interactive changing rates introduces under-sampling artifacts of the environment map during motion, which can be compensated if the change stops, by using an adaptively higher resolution for the visibility map.

Per-Cluster Matrix Factorization

In Section 2.1.18 the Bi-Scale Radiance Transfer method for image-based lighting of models covered with uncompressed BTFs was reviewed. To remove the limitation on the BTF resolution – both in the angular and spatial domain – Müller et al. [2004b] presented a combination of local PCA compressed BTFs with image based lighting.

Data Representation Similar to Sloan et al. [2003a] they compute Radiance Transfer Textures but encode them using the local PCA method [Kambhatla, N. & Leen, T.K. 1997]. In addition, similar to Sloan et al. [2003b], they apply local PCA to the transfer matrices of the mesh vertices. An approximate solution to equation 2.24 can now be computed by the weighted sum of dot products between the PCA-factors of the RTT and the transfer matrices.

Real-Time Rendering Since the dot products remain constant as long as only the camera is moving in the scene (that is, neither the mesh nor the environment nor the BTF is changing), they can efficiently be precomputed on the CPU and afterwards be stored in a texture. Since precomputation and upload times of the dot products do not allow interactive rendering for very high quality settings, the number of RTT and transfer matrix components is reduced in dynamic situations. These reductions in quality do not significantly influence the perceived quality of the rendered results as long as high quality solutions are presented in static cases.

Like in Sloan et al. [2003b], rendering requires clusters of triangles to be rendered independently, where a triangle belongs to a certain cluster if at least one of its vertices belongs to the respective clusters. This increases the rendering time slightly but the overhead is negligible for smooth meshes.

Most rendering power is spent in the fragment program which computes the weighted sum. Interpolation of the angular domain of the BTF can be achieved by standard filtering features, spatial interpolation and filtering can be achieved using standard multisampling and MIP-mapping of the BTF.

Discussion

Almost all real-time BTF rendering methods pose a great challenge to the current graphics boards and the performance differences vary greatly depending on the board and driver versions. Therefore, a rigorous comparison of the different rendering methods using for example frame-rates seems currently not possible. Instead, some general *hardware-independent* properties of the algorithms are given now, which might help judging the strengths and weaknesses of the methods.

The method of McAllister et al. [2002] is mainly suited for nearly flat and specular materials with spatially varying reflection properties. Since it only uses slightly more memory than ordinary diffuse textures and can be rendered fast, it fits into current rendering systems. This is not true for the methods which use additional data to capture also depth-varying BTFs such as [Daubert *et al.* 2001; Meseth *et al.* 2004b; Filip & Haindl 2004]. Their memory consumption and increased rendering cost restricts them to special domains and as pointed out in Section 2.1.10 the visual quality of the used analytical models still remains questionable.

Far better visual quality is offered by the methods based on matrix factorization. But using PCA alone as done by Liu et al. [2004] is only real-time for very few terms, and thus only applicable for simple materials. Therefore, a segmentation of the data into subsets may be necessary. Using a segmentation per fixed view direction as done by Sattler et al. [2003] provides excellent visual quality but requires too much texture memory to be used in complex scenes with many materials. Spatial clustering as in the method of Mueller et al. [2003] or [Suykens *et al.* 2003] reduces the memory requirements drastically while retaining high-quality but these methods face the not sufficiently solved problem of spatial interpolation and Mip-Mapping. This can result in decreased quality for texture magnification and minification. A problem of all matrix factorization based methods is the required random access to many textures, which can form a bottleneck on current graphics architectures.

2.1.19 Hierarchy

Object detail varies at many scales, as it is shown in Figure 2.25. Three main distinctions are commonly made: *macroscopic, mesoscopic* and *microscopic*. The figure shows a slightly modified version of the *hierarchy of detail* as introduced in [House & Breen 2000].

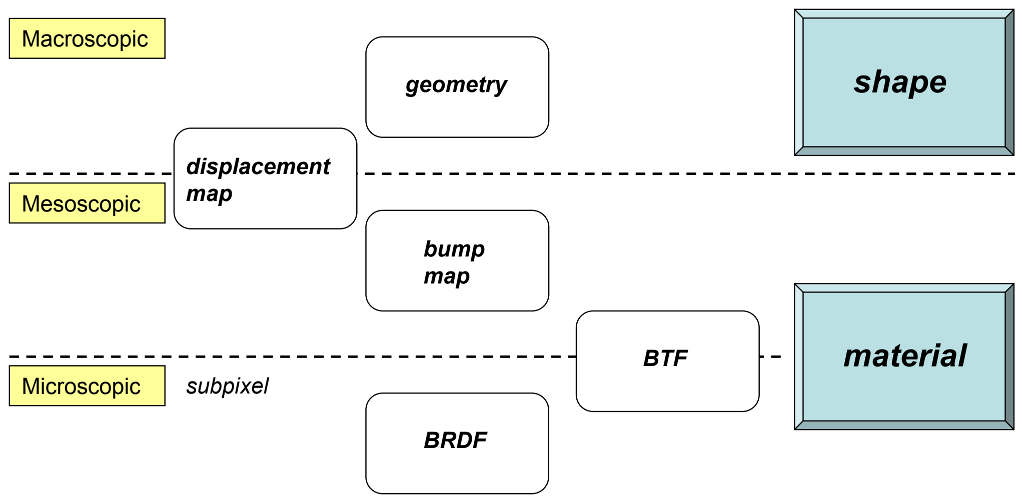


Fig. 2.25: Hierarchy of object details.

Starting at the coarsest (macroscopic) scale, the main geometry defines the shape of the object. Additional geometric details might be generated on the fly by 2D displacement maps (see Section 2.10). At the next finer level (mesoscopic), material surface properties come into effect. Besides the displacement maps, no explicit geometry is present at this level and all information is commonly stored in image textures. The simplest form is the 2D bump or normal map (see Section 2.10), which only alters per-pixel normals. In contrast to this, the 6D BTF (bidirectional texture function) (see Section2.1.6) stores on the one side subpixel surface-variant BRDF information, but also includes mesoscopic details, like self-occlusion in the material, interreflections, subsurface-scattering effects and so on.

At the finest level (microscopic), surface variations are subpixel. For uniform materials, the reflection properties can be described by the 4D BRDF (bidirectional reflectance distribution function) (see Section 2.1.7). or 6D SBRDF for surface variant BRDFs. The SBRDF is comparable to the BTF, while not capturing certain effects like self-shadowing in the material.

More advanced reflection functions, which include for instance subsurface scattering are shown in the overview chart in Figure 2.6.

2.1.20 Radiance Data

Real world radiance data, as for example shown in Figure 2.28 covers a large range of radiances values or exposures differences. If all radiance values are mapped into the RGB space, Table 2.5 shows the large difference between the average pixel values and the brightest spot at the location of the sun.

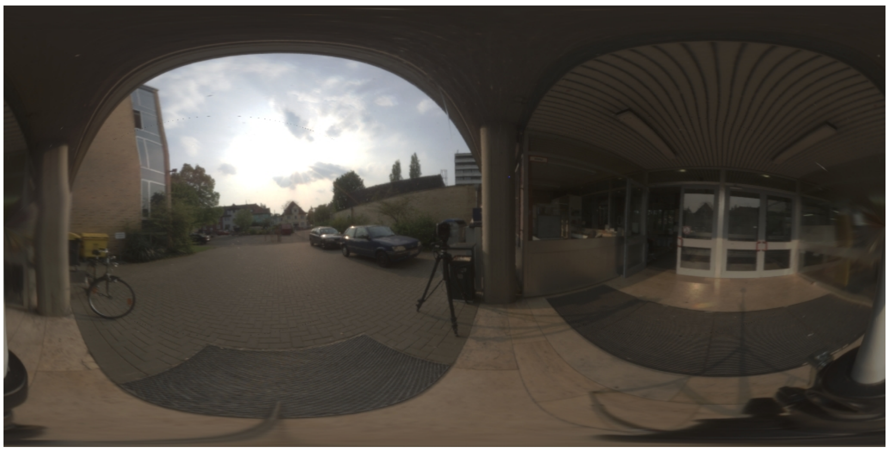


Fig. 2.26: Real world radiance data example.

	R	G	B
∅	1.3	1.2	1.0
sun	2648.0	3640.0	2280.0

Tab. 2.5: Real world radiance data example values.

Therefore, the *dynamic range D* is defined as the ratio of the highest (lightest) signal I_{max} to the lowest (darkest) signal I_{min}.

$$D = log\frac{I_{max}}{I_{min}} \tag{2.41}$$

There are several stages between the real-world radiance data and the output device, as illustrated in Figure 2.27.
In case of radiance data, beginning on the left side with real-world data, some sort of *capture device* (for example a laser scanner or a digital camera) is used to measure the data. Therefore, the analog signal is commonly converted into a digital one and further processed and stored in a specific *internal data format*. In the end,

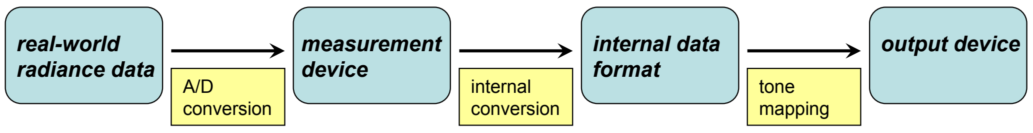


Fig. 2.27: Radiance data processing pipeline.

tone mapping (details are further down) is used to transfer the data to an *output device* (usually a monitor or TFT). Of course the measured data can be any kind of signal, for example sounds.

Due to the several stages of processing, it is important to clarify in which stage the specific data is used.

There exist a vast range of applications in computer graphics, where HDR data is necessary or will drastically enhance the visual quality. A list of applications might include the following:

- physically-based visualization
- special effects for movies or commercials
- digital film and compositing
- human vision and psychophysics
- remote sensing

In computer graphics, *High Dynamic Range Imaging* (HDRI) is the synonym for digital images with far greater dynamic range of exposures than normal (*low dynamic range* (LDR)) images. It is customary, that more than 8-bit information per usable channel is called HDR. That corresponds to a contrast ration of 255:1, in contrast to real-world values up to 100.000:1 for sunlit scenes or scenes with shiny material reflections. The 8-bit limitation is based on the historical fact, that most displays or printed media are limited at their capabilities by their very nature. Fortunately, HDR display begin to emerge [Seetzen *et al.* 2004].

There exist several formats to store HDR data. Ward and Shakespeare [1998] first introduced the *radiance format*. Amongst others, common formats are:

- Radiance RGBE Encoding

- PIXAR Log Encoding
- SGI LogLuv
- ILM OpenEXR [OpenEXR 2006]
- Microsoft/HP scRGB Encoding

These formats differ in the orders of magnitude, that is the dynamic range, they cover and therefore their accuracy in which the original data is maintained. Due to the properties of the human visual system, which is capable to distinguish luminance in a range of 10.000:1 in a single view, most often 16-bit (*half precision*) or 32-bit floating point numbers are used to represent HDR data.

Other mile stone publications in computer graphics regarding HDR usage include [Debevec & Malik 1997; Debevec 1998; Debevec *et al.* 2003]. A common technique to capture HDR scenes with LDR devices, like consumer digital cameras is to photograph a specific scenes with different exposure levels and using calibration data and the response curve of the camera to combine all LDR images into a single HDR one (see also [HDRShop 2005]).

Display of Radiance Values

The raw radiance data has to be fitted to a certain limited luminance range of the output device, which in most cases is a LDR one. The process of fitting the data to the screen *gamut*, that is a subset of colors, which can be accurately represented, is called *tone mapping*.
There exist several different *mapping operators* which can be roughly divided into the following groups [Debevec *et al.* 2004; Meseth *et al.* 2004a]:

- based on image formation
  - frequency-based
  - gradient-domain
- based on human vision
  - global operators
  - local operators

Here, depending on the desired output, the complexity of the operator will change. For several applications (for example *gaming*), also real-time performance is desired.

Another, historical enforced technique to display radiance values, is *gamma correction*. Due to the nonlinear relationship between the voltage input and the light output of a *Cathode-Ray Tube* (CRT) (see also [Akenine-Möller & Haines 2002]), each color value has to be adjusted accordingly to match the perceived brightness:

$$c = c_i^{1/\gamma} \tag{2.42}$$

where c_i is the input color and γ is device specific (for example 2.3-2.6).

2.1.21 Illumination

Scene illumination is a key element for rendering realistic images. In a nutshell:

> *illumination calculation is visibility determination*

That is, the algorithm has to determine, whether a specific point in space is visible (and being lit) by parts of or the complete light source, or not (and lies in shadow).

In addition to that:

> *illumination calculation is relative position in space determination*

That is, the algorithm has to determine the relative positions in space of the light source and the receiver to each other. This is necessary to be able to apply the correct illumination or shading model (see also Section 2.1.4 and 2.1.8).

To accomplish this task efficiently it is common to use some approximations. On the one hand the usable type of light source may be limited or as with some of the shading models, visibility determination is skipped completely.

Light Source Types

Several light source types exist in computer graphics:

- point light sources
- directional light sources
- area or extended light sources

Point or *directional* light sources are the simplest form of light sources imaginable. Defined only by a position or direction in space, they are most suitable for simple ray geometry calculations. In addition and based on the OpenGL standard, shading operations with them are hardware-supported. On the other hand they have no real-world counterpart and therefore it is hard to achieve visual pleasing results using only this type of light source.

Area or *extended* light sources closely resemble real-world lighting. On the other hand, while not directly hardware-supported, algorithms which can handle this type of light sources are usually hard to perform due to their computational complexity if both, visibility and position determination are done for all parts of the light source.

Environment Illumination

Environment or *reflection mapping* as introduced by Blinn and Newell [1976] is a technique to approximative generate reflections of the environment on a curved surface. It is based on computation of a reflection vector and a transformation into spherical coordinates.

Miller and Hoffmann [1984] introduced *sphere mapping*. Here, the environment is stored in a perfectly reflective sphere, the *light probe*. An example of this is shown in Figure 2.28.

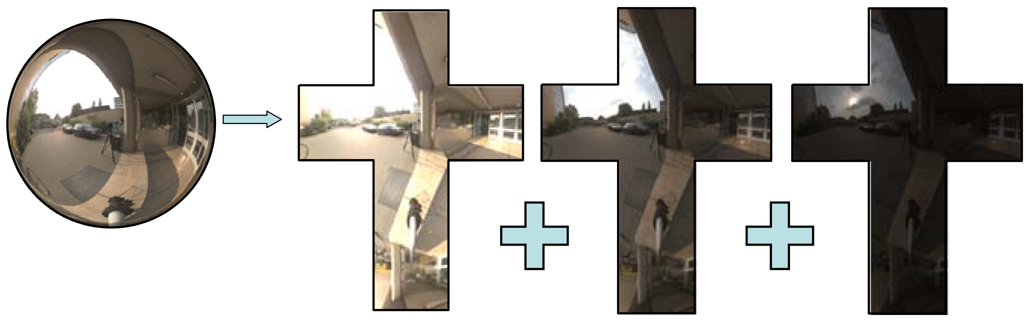


Fig. 2.28: High-dynamic range assembly with LDR images.

The measurement and processing of *environment illumination* is illustrated in Figure 2.28 and 2.29. The probe is used to capture the illumination at a specific point. Using multiple exposure times, HDR data can be generated. With a program like HDRshop [HDRShop 2005] the images can be transformed and integrated into a so called *cube map*.

In contrast the original environment mapping, Greene [1986] uses a *cubic* environment map. An example is shown in Figure 2.29. The creation process and storage is far easier than the original method and it is well suitable for hardware acceleration.

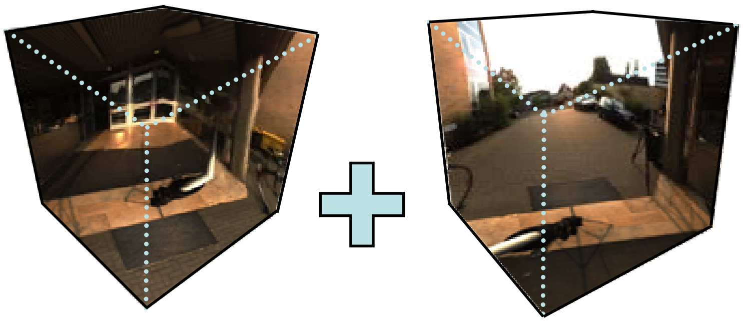


Fig. 2.29: Illumination stored in a cubic environment map.

As an variation on sphere mapping, *paraboloid mapping* was proposed by Heidrich and Seidel [1998]. Here, two paraboloids are used to store the reflections of the environment. As an advantage, this method generates no singularity and reflection interpolation is possible. Therefore, it is view-independent and can also be hardware-supported.

More details about the discussed methods can be found for example in [Akenine-Möller & Haines 2002].

2.2 Geometry Processing Techniques

This section briefly describes the geometry processing techniques *Mesh Simplification* and the *Level of Detail* concept, which are used in Section 4.2.1 (*Shadows: Perception*) and *Silhouette Edge Detection* used throughout Chapter 4 (*Shadows*).

2.2.1 Mesh Simplification

The main goal of polygonal mesh simplification is the generation of a new mesh $\mathcal{M}_{n-1}$ containing less geometry, than the original mesh $\mathcal{M}_n$, while resembling a similar appearance. This procedure is carried out until the coarsest mesh $\mathcal{M}_0$.

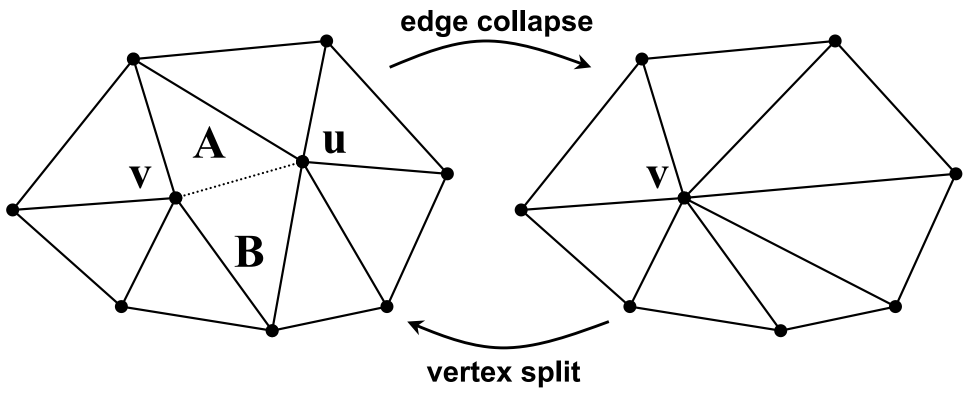


Fig. 2.30: Mesh simplification through *edge collapse* and the reverse operation *vertex split*.

Several simplification *operators* exist, which allow the geometry reduction. The well known *edge collapse* and its reverse operation *vertex split* are illustrated in figure 2.30. The edge $\mathbf{uv}$ of the mesh $\mathcal{M}_n$ is collapsed into the point $\mathbf{v}$, thereby removing the shown triangles **A** and **B** and resulting in mesh $\mathcal{M}_{n-1}$.
The *vertex merge* operator is shown in figure 2.31. The two vertices $\mathbf{v_1}$ and $\mathbf{v_2}$ are merged into a single new one $\mathbf{v_3}$.

In addition to this, Borodin et al. [2003] have produced high-quality results by combining generalized pair contractions and extension of the vertex merge operation. In 2005, the computationally expensive dense regular sampling was replaced with a significantly faster adaptive sampling method by Guthe et al. [2005].

Although through the usage of the simplification operators the geometry is reduced, the quality of such an operation has to be determined. That is, the error

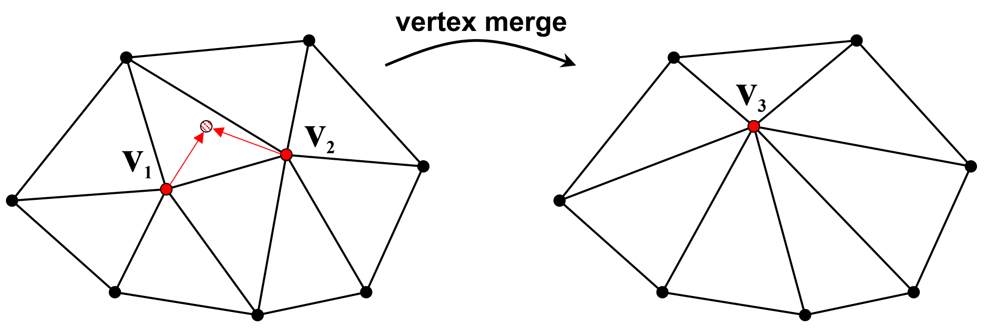


Fig. 2.31: Mesh simplification through *vertex merge*.

introduced by the operation is measured and compared against other errors introduced by alternative operations. All possible local operations are performed virtually and each error is determined. The operation with the least error compared to the allowed error is chosen. Several error measurements exist, using metrics based on viewer position, object features, object volumes etc. The complexity of the chosen error measurement significantly influences the processing speed.
As a measurement criterium for the simplification, the Hausdorff[1]-Error can be used. The Hausdorff-Error describes the geometrical distance between the original and the simplified mesh and vice versa. Klein et al. [1996] first used it to control the simplification.

For further reading, [Puppo & Scopigno 1997] and Luebke et al. [2001] give a detailed review of simplification algorithms and links to relevant literature.

The reverse process of simplification is called *refinement* as also shown in Figure 2.30 as vertex split. Different *subdivision* schemes exit, for example [Loop 1987].

2.2.2 Level of Detail

Using the above techniques for mesh simplification, several *Level of Details* (LODs) can be generated out of a given object. That is, several versions of a geometric object with decreasing number of faces and vertices.

An example is illustrated in Figure 2.32. Here, the top row shows the *Stanford Bunny* in the original (70k faces) and simplified versions (2k and 0.2k faces) as wireframe renderings. The middle row shows the shaded versions.

[1]Felix Haussdorff, mathematician

The bottom row shows the common usage of several LODs. Assuming that the objects moves away from the user, it is possible to use increasing LOD without loosing the visual appearance of the object. On the other hand, rendering speed is increased, because of the lower number of geometry to be drawn and shaded.

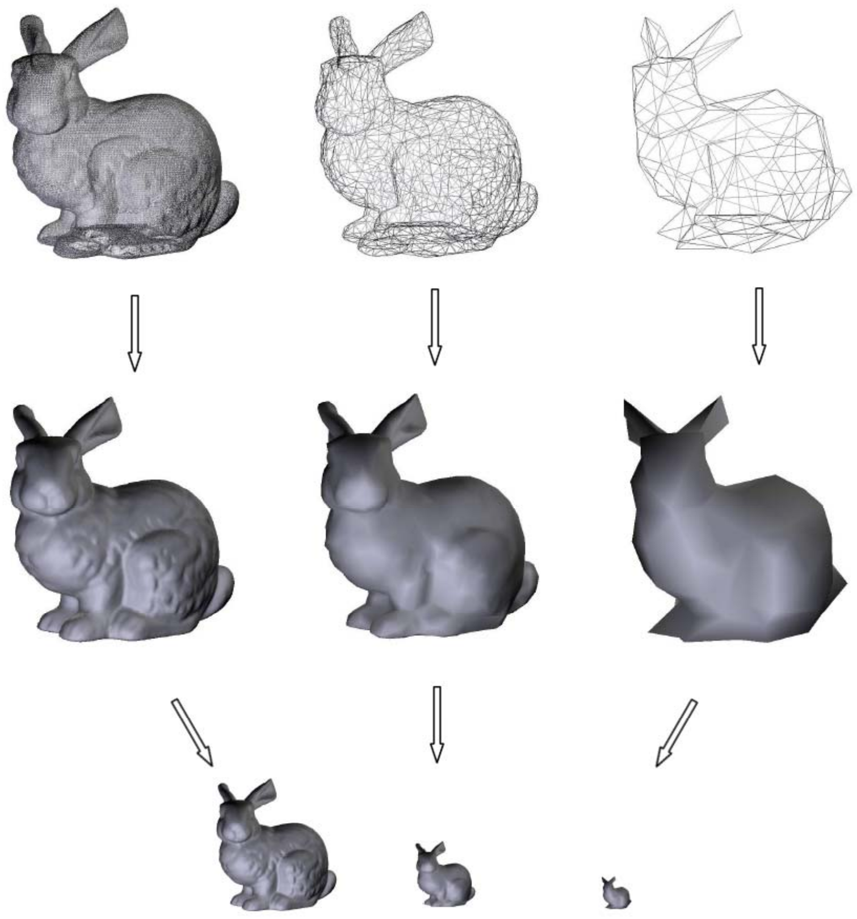


Fig. 2.32: Example for a LOD generation.

2.2.3 Silhouette Edge Detection

Silhouette detection in a rendered image is needed in different areas of computer graphics. Non-Photorealistic Rendering (NPR) for example makes heavy use of silhouettes and also some shadow rendering algorithms need the detection of silhouettes (*Shadow Volumes*, see 2.4.1).

To detect silhouette edges, two main approaches exist. The *image* based approach renders the geometry in such a way, that the silhouettes fall out during rendering, for example through efficient rendering of front -and backfacing polygons.

Another image based approach is to analyze the image after it is rendered. The extraction of the silhouettes (which is done like edge detection) can be supported by the rendering, for example via depth images, color coding, object-IDs or special shading. This kind of approach can easily be done within a shader program (see 2.3.3) on the GPU. Using image based approaches, most of the time no geometrical information about the silhouette edges is available.

The second group tries to find all edges geometrically, that is *object* based. While this approach might involve a significant computational load, it gives the most flexibility in how silhouettes are handled after detection.

Assuming a polygonal model with adjacency information, silhouette edges are defined as follows.

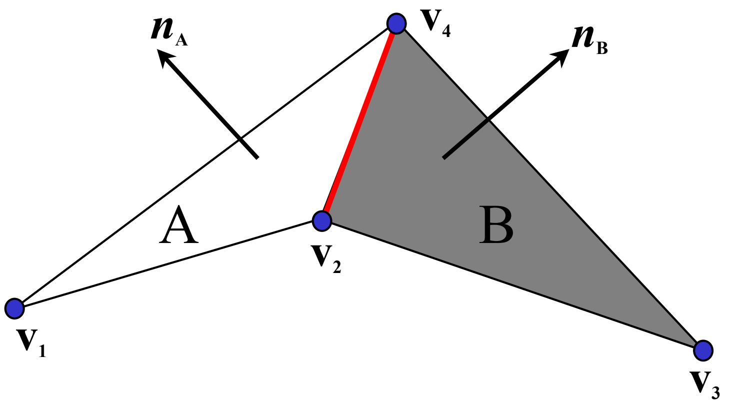


Fig. 2.33: Silhouette edge (*red line*) from v_2 to v_4 with adjacent faces A and B.

As shown in Figure 2.33 the two triangles **A** and **B** with the face normals n_A and n_B respectively, share the edge v_2v_4. This edge is a silhouette edge, if the two normals face different directions in respect to a defined viewer position (towards/away from the viewer).

Per definition, *border edges*, that are edges which are only belong to one single triangle, are also silhouette edges.

The brute force way, that is to check every edge every frame the viewer position is changed, might be very time consuming. Markosian et al. [1997] have proposed a method to speed up this process. They mainly use frame-to-frame coherence and a statistical approach to detect most of the silhouette edges fast. Other methods propose to use simplified versions of the original mesh to do the detection process [Kirsanov *et al.* 2003]. The center image in Figure 2.34 illustrates the brute force way.

The detection process can also be ported directly onto the GPU as shown for example in [Brabec & Seidel 2003; McGuire & Hughes 2004]. One possibility to detect the silhouette edges is shown on the left image in Figure 2.34. Here, hardware supported *Occlusion Queries* (2.3.4) are used to determine visibility of triangles, therefore automatically detect also the silhouette edges.

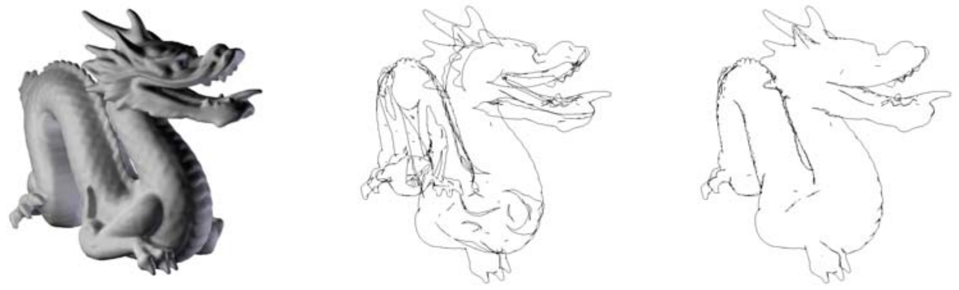


Fig. 2.34: Silhouette Edge Detection example for the *dragon* model.

For further reading, a good overview article is Isenberg et al. [2003].

2.3 Graphics Hardware

2.3.1 Overview

The evolution of computer graphics hardware started back in the beginning of the sixties. The first graphic output devices were called *vector displays*. They simply stored point- and line-plotting commands, which were interpreted by a *vector generator*, which finally modified an electron beam accordingly for the direct screen output. In 1963, Sketchpad was introduced by Sutherland [1963]. Using a light pen device, interactive design with a vector display was possible.

In the mid 80´s the Video Graphics Array (VGA) card was invented at IBM. With the decrease of memory prizes, these *raster displays* showed up. The drawing commands now were interpreted and converted into a raster, which simply was a piece of memory. Therefore, the term *rasterization* was born. The memory, called display buffer, was arranged in form of a matrix. The matrix cells represented the entire visible screen and were scanned out sequentially by a video controller before the output on a screen. The matrix cells were called picture elements or *pixels*. At this time, Silicon Graphics (SGI) workstations that supported real-time raster line drawing were state-of-the-art.

At the beginning of the 90´s of the last century, specialist hardware from SGI had 24-bit raster display and hardware support for interpolated Gouraud shading and depth buffers (2.3.2). Due to the difficulties with different hardware types and their Application Programming Interfaces (APIs) and the time consuming programming process, a standard programming interface emerged in 1992.

Based on SGI´s IrisGL, the Open Graphics Language [OpenGL 2005] API was created by the Architecture Review Board (ARB), a network of hardware and software manufacturers led by SGI. In contrast to its predecessor, OpenGL does not rely on any hardware support of its functions. Therefore, as long as the code is written against the specification, the program will run on any graphics card with OpenGL support, regardless if it provides hardware or software driver support for the used function.

As stated in [Neider *et al.* 1997], "*OpenGL is really a hardware-independent specification of a programming interface. You use a particular implementation of it on a particular kind of hardware.*"

In 1995, Microsoft released his own API, called D3D, which later was renamed into DirectX. DirectX does not only include graphics, but also standards for sound or network programming and input devices. It does only work with the Windows operating system.

Beginning with this year and the success of the personal computer in the mass market, the first graphic *accelerator* cards emerged. The growing game and entertainment market was the driving power behind development since then, culminating in a new hardware generation each six months in nowadays.

The evolution is closely related to Figure 2.35, showing the general graphics pipeline. This pipeline is mainly based on the internal OpenGL rendering pipeline. The first graphic cards only integrated parts of the rasterization stage on the hardware. 3Dfx (now part of NVidia) and their Voodoo 2 card were the first to have the complete triangle setup on the card. While the first cards only supported native APIs, the RIVA chipset by NVidia was the first one to fully support OpenGL. Parts of the transform & lighting stage (T&L) first appeared in 1999 on a graphics chip in form of the GeForce 256 by NVidia.

Since then, graphics memory sizes increased and more and more functionality made it onto the graphics board. Along with this, several extensions were invented by the major manufactures and in parts also integrated into the OpenGL and DirectX specifications. That is, the APIs and the hardware influenced each other in their evolution.

2.3.2 Architecture

Figure 2.35 shows the general architecture of a state-of-the-art graphics processing unit (GPU) and its relation to the central processing unit (CPU). GPUs are highly parallel streaming processors optimized for vector operations, with both MIMD (multiple instructions multiple data) and SIMD (single instruction multiple data) pipelines in the vertex and in the rasterization part, respectively.

The memory bus between the CPU and the GPU is one of the possible bottlenecks of a graphics application. Depending on the used algorithm, data has to be processed on the CPU first, before it could be sent to the GPU. If this data amount is too large, the application might be bus width limited.

The GPU itself is mainly split up into two processing stages. The first is the *Geometry Processing* stage. The second is the *Rasterization* stage. The hardware is optimized for serial data flow and the processing of huge amounts of simple shaded primitives (for example triangles). There exist possibilities to send data *against* the pipeline or back to the CPU, but this normally involves the loss of speed. Each

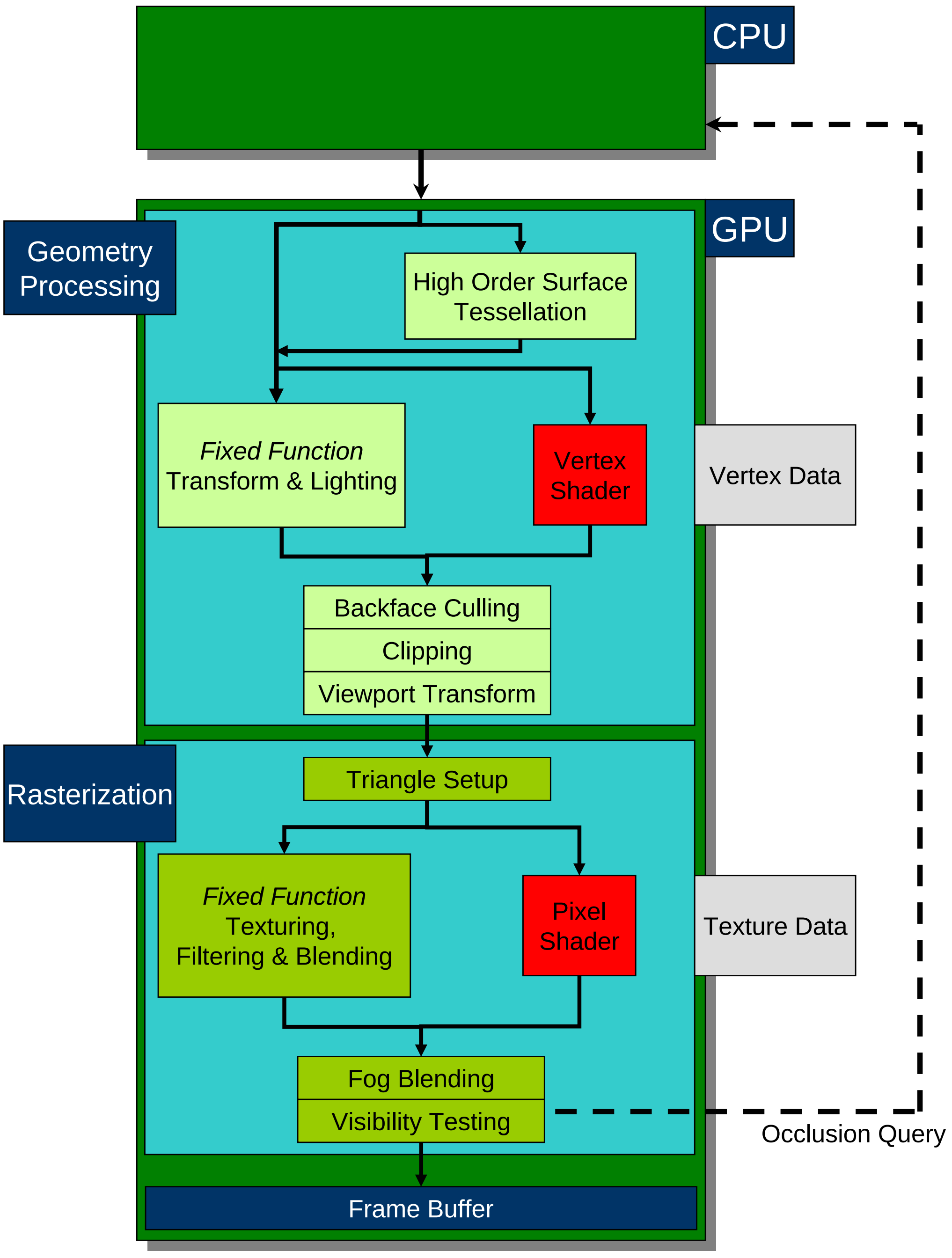


Fig. 2.35: Graphics pipeline overview.

stage has access to certain areas of memory, namely *vertex data* and *texture data* to store intermediate results or data from the CPU.

Main purpose of the first stage is to calculate the *transform* and *lighting* (T&L) aspects of the primitives. That is, to geometrically orientate the vertices of the primitives in space and do vertex based lighting calculations. There are special functions available to tesselate high order surfaces like Bezier Surfaces on the fly from control points first.

To reduce the computational load in the later stage, basic visibility calculations are also performed. All surfaces which point away from the viewer are *culled* and all surfaces which lie outside the visible screen space are *clipped*. Finally, to map the scene onto the used window size, *viewport transformation* is used.

In the second stage, the mapping of the remains of the geometry into the display buffer is performed. This is done via the *triangle setup*. Here, the rasterization generates so-called *fragments* out of the geometry data. At this point, it is not cle-ar if the fragment finally gets rendered on the screen to become a pixel, because several following stages, for example the visibility test, can *kill* the fragment.

To further enhance visual quality of the fragment, it can be *textured*, *filtered* and *blended* with several functions defined by the OpenGL API.

The next substage is the addition of *fog blending*. Due to performance reasons, graphic applications tend to limit the distance the viewer could look at. Therefore, all parts of the scene which are further away, will be covered by distance fog and not be rendered.

The last stage before writing the fragment as a pixel into the *frame buffer* (see Section 2.3.2), is the *visibility test*. Here, the depth value of the fragment is compared to the existing depth value in the *depth buffer*, which is explained in Section 2.3.2.

After the visibility test is passed the fragment finally becomes a pixel. This information can be efficiently read back by an *occlusion query*, which is described in detail in section 2.3.4.

Most substages are optional along the pipeline and can be switched on or off during rendering, allowing for speed up certain algorithms. Some stages (fixed functions) can completely be replaced by programmable *shaders* (see Section 2.3.3).

Modern graphics cards also integrate several *rendering pipelines*. This means, parts of the rasterization stage are performed in parallel on several fragments at once.

The future of graphic cards will definitely be an enhanced programmability. That is, all parts of both stages will be completely reprogrammable. In addition, the two main stages might be merged together to allow for completely new algorithms to be implemented entirely on the GPU. Therefore, an efficient memory and pipeline management will be the greatest challenge for the engineers.

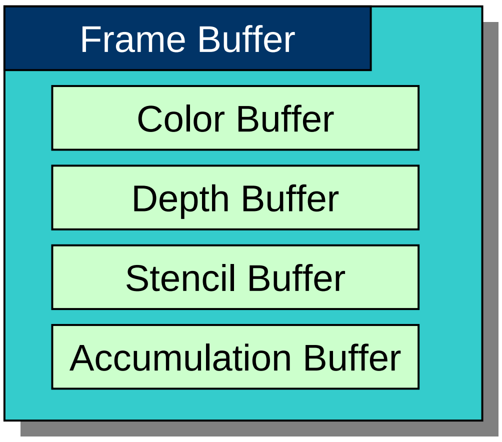


Fig. 2.36: Frame buffer organization.

The *frame buffer* (see also Figure 2.35) itself consists out of four main parts, as shown in Figure 2.36. These parts are called buffers and each buffer is organized as a 2-dimensional matrix. The size of the buffers is only limited by the available graphics memory, but is at least as large as the screen resolution to be displayed [Neider *et al.* 1997].

The *color buffer* is the most important one. For each pixel it stores four 8-bit values, which are called RGBA. This stands for **R**ed, **G**reen, **B**lue and **A**lpha. The first three values simply represent the color of the pixel, the last is commonly used as a transparency value. To allow for flicker-free animation *double buffering* is used. That is, two color buffers (front and back) are switched each frame. One is used for display, and the other is used for writing the new image. For stereo display this method is extended to *quad buffering*, which is simply the addition of two more buffers (left and right).

The *depth buffer*, which often is also called Z-buffer, contains depth information for each pixel position. Depth is usually measured in terms of distance to the viewing position. Smaller values will overwrite larger ones. That is, after rendering the depth buffer contains pixel-wise information to the closest objects in respect to the viewer. Using the so called *depth-test* operation, these values can for example be used to control further rendering.

The *stencil buffer* is used for masking. It contains 1-bit values for each pixel position, which are used to allow or restrict writing color information into the color buffer for the specific position. This can be used to prevent certain screen areas from overwriting.

The last buffer is the *accumulation buffer*. It is organized similar to the color buffer, containing RGBA values. The main purpose is to accumulate a series of images into one final one. This feature can be used for certain post-processing effects like motion-blur or anti-aliasing. Direct writing is not possible, only copying between the accumulation and the color buffer.

2.3.3 Shader Programming

With the emerge of new generations of graphics hardware since the beginning of the new millennium, the programmability increased in large steps. As illustrated in Figure 2.35, the fixed function parts in *Geometry Processing* and *Rasterization* are replaceable by so called *shaders* (colored in red). These shaders are operated by little shader programs, consisting out of a certain set of basic instructions, which are executed each time the stages are passed.

```
varying vec3 normal;
varying vec3 vertex_to_light_vector;

void main()
{
        // Defining The Material Colors
        const vec4 AmbientColor = vec4(0.1, 0.0, 0.0, 1.0);
        const vec4 DiffuseColor = vec4(1.0, 0.0, 0.0, 1.0);

        // Scaling The Input Vector To Length 1
        vec3 normalized_normal = normalize(normal);
        vec3 normalized_vertex_to_light_vector = normalize(vertex_to_light_vector);

        // Calculating The Diffuse Term And Clamping It To [0;1]
        float DiffuseTerm = clamp(dot(normal, vertex_to_light_vector), 0.0, 1.0);

        // Calculating The Final Color
        gl_FragColor = AmbientColor + DiffuseColor * DiffuseTerm;
}
```

Listing 2.1: example GLSL Fragment Shader

The shading languages are similar to the C programming language [Rost 2004], supporting loops and branching, including if, else, if/else, for, do-while, break, continue, etc. Listing 2.1 shows an example fragment shader for simple lighting computation. Using the *ARB_fragment_shader* extension [OSS 2005], this shader is loaded onto the graphics card.

The most used representatives of shading languages are GLSL (Graphics Library Shading Language) [GLSL 2006] and CG by NVidia [CG 2006]. By now, GLSL is included into the OpenGL 2.0 core. Benefits of using GLSL are its cross platform compatibility on multiple operating systems and cross hardware platform compatibility on any card which supports GLSL. In addition, each hardware manufacturer can exploit optimized code for their particular graphics cards architecture in the driver.

A kind of predecessor to the high level languages were the shader programs, which were used with the *ARB_fragment_program* extension [OSS 2005]. An example is shown in Listing 2.2. The programs are simple text files, containing assembly-like instructions like MUL (multiply) or TEX (texture lookup). Coding and debugging these programs is time-consuming, therefore the new shading languages described above are going to replace this kind of graphic card programming.

```
!!ARBfp1.0
# texturing with HDR environment

OUTPUT output=result.color;
TEMP component00;

TEX component00,fragment.texcoord[0],texture[0],2D;

MUL component00,fragment.texcoord[1],component00;
MUL output,fragment.texcoord[2].x,component00;
END
```

Listing 2.2: example pixel shader code

2.3.4 Occlusion Queries

Occlusion Queries were first proposed by HP [OSS 2005], to be integrated into OpenGL. As shown in Figure 2.35 they provide a way to get information from the GPU back to the CPU after all stages of the rendering pipeline. Their typical usage is for visibility testing and culling.
The first version of occlusion queries worked the following way. First, an occlusion query counter is switched on, then geometry is rendered. After the counter is switched off, a boolean flag is send back to the CPU, stating if the depth buffer

was modified during the rendering, that is if the geometry was rendered at all.

Therefore, if a complex scene is present, efficient culling is possible. A bounding box of a certain part of the geometry is rendered first. If the query test fails, all geometry inside the box is skipped. Several algorithms use this form of occlusion queries, for example [Staneker *et al.* 2003; Staneker 2003; Bittner *et al.* 2004].

The HP version of the query uses a stop-and-wait execution model. That is, it is not capable of doing multiple tests in parallel. Therefore, the latest OpenGL version of the occlusion query [OSS 2005] now provides two improvements. First, not only a boolean information is sent back, but the exact number of rendered pixels. And second, it is now possible to issue several queries at once and exploit the parallelism between CPU and GPU.

Other examples for the usage of occlusion queries are Order Independent Transparency (OIT) [Rege 2002] or shadow calculations [Sattler *et al.* 2004a] (see also Section 4.4.1).

2.4 Shadows

While in nature shadows are simply the absence of light (*photons*), in computer graphics *calculating* shadows is commonly a hard problem.

In essence, all shadow algorithms are visible-surface detection algorithms. That is, they somehow calculate, whether a surface point has a direct line-of-sight to a light source. If multiple light sources are present, the classification must be done relative to each of them. Those points, which have none direct light-of-sight are in complete shadow.

Photon Mapping as described in [Jensen 2001] is one of the techniques, which comes closest to the physical understanding of light transport by calculating paths of a vast amount of virtual photons through the scene. This approach is time consuming and requires efficient memory handling.

Mainly three parts are concerned, when calculating shadows. At first, the light source itself, which illuminates the scene. The emitted light first hits the *occluder*, which blocks this light from hitting the *receiver*. Occluders and receivers can be different objects or fragments of objects down to geometric primitives. Of course it is possible, that a receiver is the occluder relative for another receiver.

But due to close relation of computer graphics to mathematics, two sorts of light sources are considered. Point light sources are treated as infinitesimal small points in space emitting light. Compared to the real world, they have no direction equivalent and have to be considered as infinite distant directional light sources. On the other hand, calculations can be done very quickly and depending on the used scene resolution, point light sources can be a reasonable approximation.

In contrast to this, area or extended light sources resemble real-world light sources like torches, light bulbs or the sun. While this light source type produces realistic shadows, no direct hardware support is available and computational complex algorithms have to be used.

Therefore, depending on the used type of light source, two main shadow types exist, as illustrated in Figure 2.37. The first, *hard shadow*, is cast by point light sources (right image), while the second, *soft shadow*, is cast by area light sources (left image). The latter is explained in Section 2.4.3. Regions, which are complete in shadow, are the shadow´s *umbra*, while regions which are only partially shadowed are the shadow´s *penumbra*.

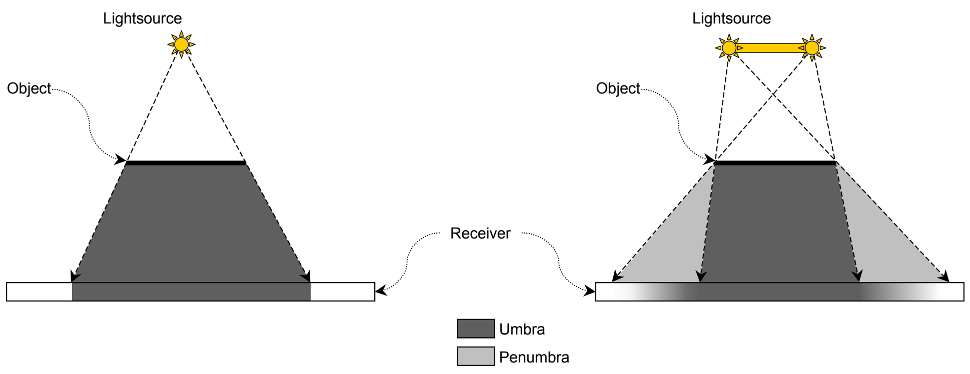


Fig. 2.37: Shadows cast by different light source types. **Right:** point light source. **Left:** extended light source.

The common graphics hardware pipeline is optimized to render a lot of simple shaded triangles fast (see Section 2.3) and has no direct support for visibility determination. Therefore, a lot of the approaches originated on the CPU and were not ported onto the GPU until the emerge of programmable shaders (see 2.3.3). In the following, the most common techniques to generate these shadows in computer graphics are described.

2.4.1 Shadow Volumes

A classic way to compute shadows for point light sources are shadow volumes as introduced by Crow [1977].

The main idea is illustrated in Figure 2.38. Beginning with a point (light source) and a blocker object (triangle), a truncated infinite pyramid is constructed (grey), which is called *shadow volume*. All points within this volume are obviously in shadow.

To evaluate, whether an arbitrary point in the scene is within the volume, a ray from the observer to this point is traced. If the ray enters a shadow volume a counter is incremented. If the ray leaves a volume, the same counter is decremented. This is done until the point is reached. If the counter is zero the point is not in shadow, otherwise it is. This process is called the *z-pass* algorithm and illustrated in Figure 2.39.

For each of the blocking objects shadow volumes have to be constructed. If there is more then one light source, this process has to be repeated accordingly.

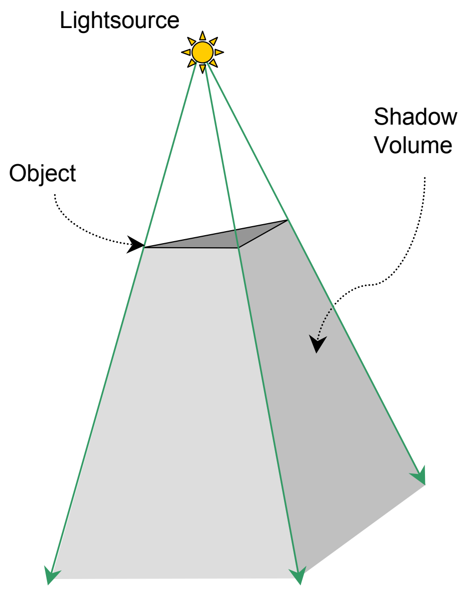


Fig. 2.38: Generating the shadow volume.

The shadow volume is defined by the position of the light source and bounded by a set of shadow polygons. Each of the quadrilateral shadow polygons is attached to one *silhouette edges* (see 2.2.3), which is found in respect to the light source. Therefore, if the light source is moving, the shadow volume has to be recalculated, that is, the new silhouette edges have to be detected.

Doing the counting geometrically is time-consuming. A smart solution is the usage of the stencil buffer (see 2.3.2) as proposed in [Heidmann 1991] or other buffers for the counting. Here, the shadow polygons are rendered into the buffer, which is adjusted automatically.

A major problem with the *z-pass* algorithm is the position of the observer. If the observer is within a shadow volume, that is, if the counting starts within a shadow volume, the result will be wrong. For correction, the fact that the observer is within a volume has to be known beforehand. Another problem is introduced by the near clipping plane of the viewing frustum used for rendering, which might intersect with one of the shadow polygons.

Bilodeau and Songy [1999] presented an alternate approach, later called *z-fail*,

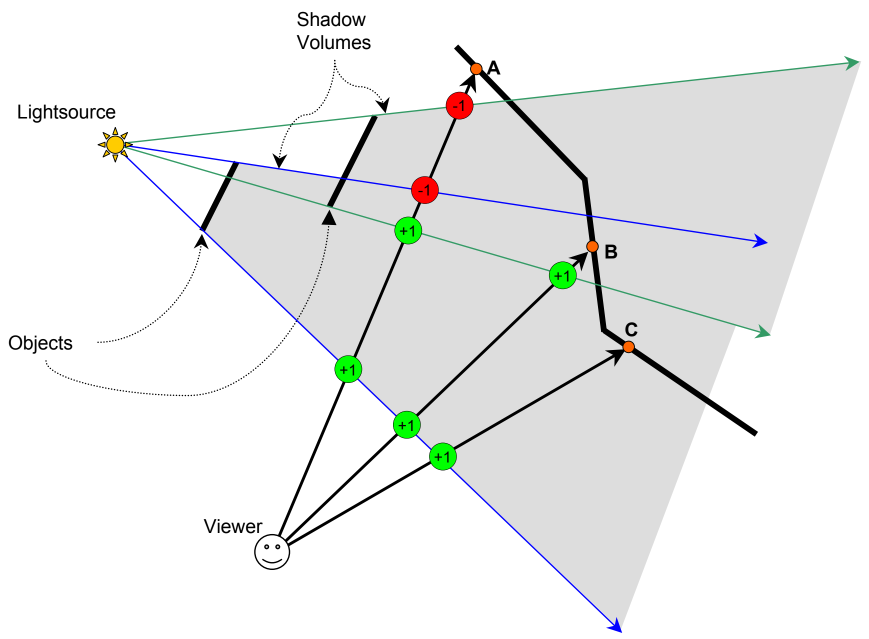


Fig. 2.39: Shadow volumes in side view; *z-pass* algorithm.

which is illustrated in Figure 2.40 to cope with these problems. The main idea is to close the infinite shadow volumes with endcaps and reverse the direction of counting, that is counting from outside in the direction of the viewer. Thus, the position of the viewer can not lead to wrong results.

The algorithm has several advantages. Because it is not image, but object based, it produces sharp shadow boundaries and has no aliasing artifacts. It also only requires a counter buffer (stencil buffer), which might even be hardware supported.

On the other hand, some disadvantages are also presented. Because of the limitation of the stencil buffer size, semitransparent objects can not be handled correctly. To reduce the complexity of the shadow volume computation, silhouette edge detection is needed. Even with this reduction, a major performance problem is the fill rate limitation of the algorithm. All shadow polygons have to be draw into the stencil buffer. An approach to reduce this, can be found in [Aila & Akenine-Möller 2004]. Independent of these limitations, several newer game engines use this technique [UNR 2006; ID 2006].

For further reading and hardware implementation details [Akenine-Möller & Haines 2002; Everitt & Kilgard 2002; Brabec & Seidel 2003] are recommended.

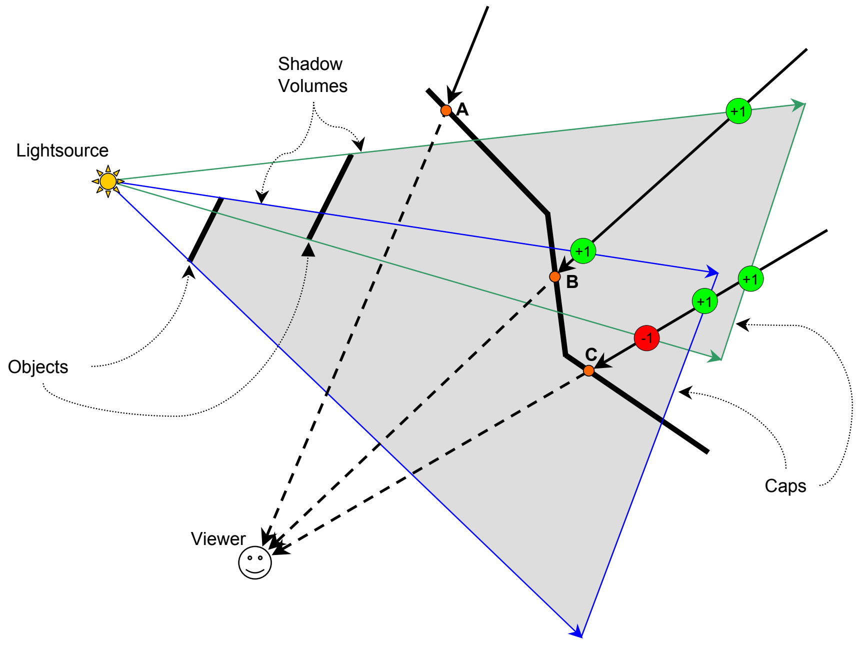


Fig. 2.40: Shadow volumes in side view; *z-fail* algorithm.

2.4.2 Shadow Maps

Shadow volumes as mentioned above have the main disadvantage, that the computational load scales with the scene complexity. In contrast to this, Williams [1978] proposed to use the Z-buffer or depth buffer (see 2.3.2) to generate shadows quickly on arbitrary objects and scene independent. The 2-pass algorithm works in image space and is described in the following.

First, the scene is rendered from the view of the light source $L(x'_0, y'_0, z'_0)$ into the depth buffer as shown in Figure 2.41. This results in a depth map (often called *shadow map*), which contains the distance of each object closest to the light source for each pixel. That is, the distances $\overline{La_1}$ and $\overline{Lb_1}$ are written at the positions a and b in the depth map, respectively.

The resulting light view and the depth buffer is shown in Figure 2.43 in the left and center image. Here, dark means closer to the viewer. The images were created with the NVIDIA shadow map demo [NVD 2006].

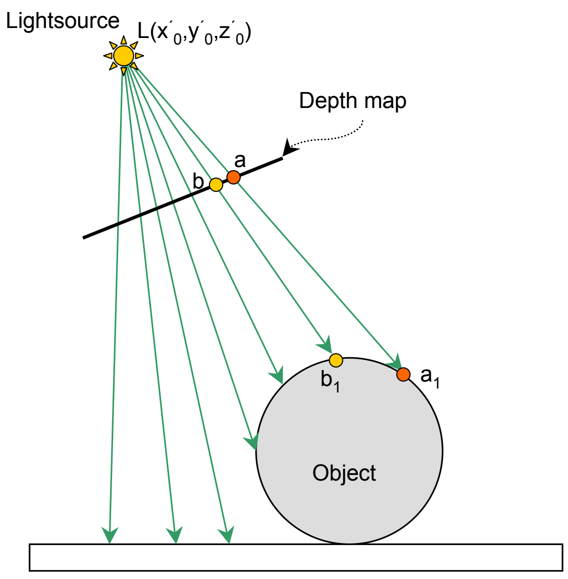


Fig. 2.41: Shadow map algorithm, first pass.

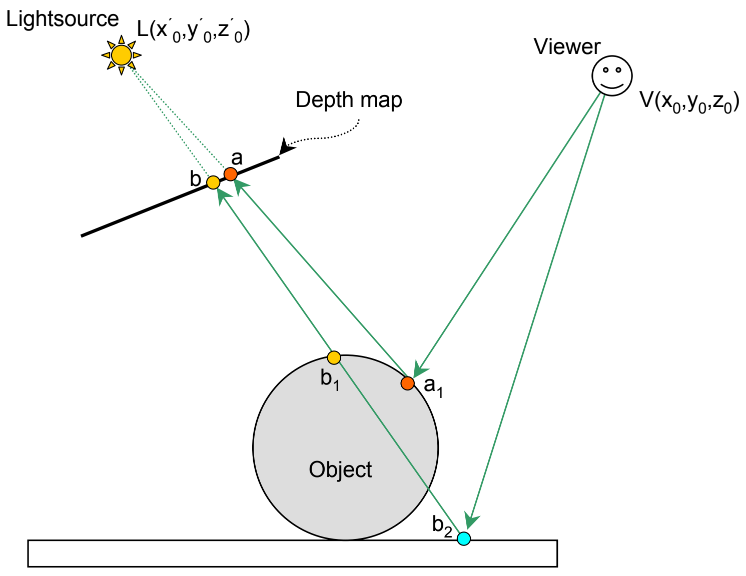


Fig. 2.42: Shadow map algorithm, second pass.

Now, the scene is rendered from the viewing position $V(x_0, y_0, z_0)$ as shown in Figure 2.42. While rendering, the position of each fragment is projected into the local coordinate system of the light source ($V \rightarrow L$). The projected z-value is now compared to the corresponding value stored in the shadow map. If the rendered fragment is farther away from L than the stored value, the point is in shadow, otherwise it is not and the final image can be rendered (see right image in Figure 2.43). For example, the distance of the projected point b_2 ($\overline{Lb'_2}$) is larger than the depth value stored at b in the depth map ($\overline{Lb_1}$), therefore b_2 is in shadow.

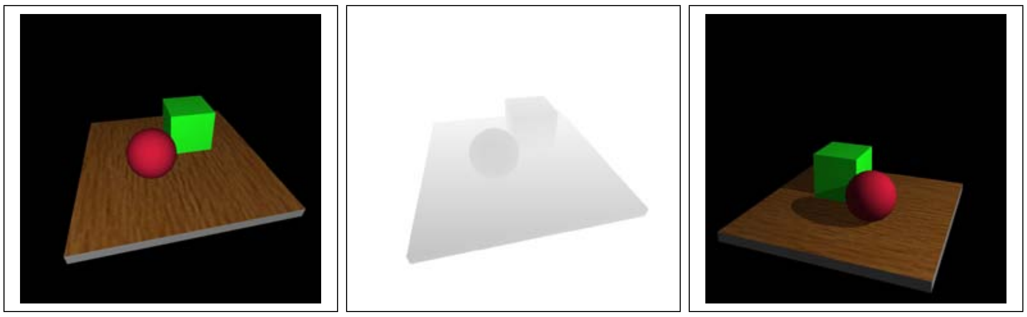


Fig. 2.43: Shadow map algorithm examples; **Left:** light view; **Center:** depth map; **Right:** eye view.

To speed up the complete process, texture mapping hardware was exploited by Segal et al. [1992] and the projection process is also hardware-supported with the *ARB_shadow* extension [OSS 2005].

The shadow map algorithm with its image-based nature naturally implies two major drawbacks, *aliasing artifacts* and *incorrect self-shadowing*. Fortunately, there exist some algorithms to cope with each of them.

The resolution of the depth map usually is fixed to a certain size. Due to the perspective view onto the scene, closer objects are larger, than objects which are farther away. This is not regarded by the shadow map. Therefore, **Perspective Aliasing** occurs. That is, near the camera, the shadow map resolution is not high enough, to avoid blocky shadow pixels.

To handle this problem *perspective shadow maps* were introduced [Stamminger & Drettakis 2002] and extended [Wimmer *et al.* 2004]. The idea here is to redistribute the shadow map pixels, to virtually increase the resolution near the camera. This is done via an additional perspective transformation into light space to generate the shadow map in normalized device coordinate space.

Projection Aliasing is another sampling problem. If the angle between the light source and an object is near the right angle, that is, if the light is almost parallel to the surface, only a few shadow map pixels are used for this surface, and the shadow stretches along that surface. This could only be solved with a local increase of the shadow map resolution. Approaches to solve this aliasing problems can be found in [Hourcade & Nicolas 1985; Reeves *et al.* 1987; Tadamura *et al.* 2001; Fernando *et al.* 2001; Aila & Laine 2004].

In a nutshell, the aliasing problems can be reduced, by redistributing the limited number of shadow map pixels in an optimal way, depending on the viewed scene configuration. This is combined with the increase of the numerical stability of the projection calculations.

At last, **incorrect Self-Shadowing** can occur. Due to numerical precision during the projection and the limited precision of the depth map, the comparison between the depth values might produce incorrect results. Therefore, Moiré patterns can be visible.

The problem is often unsatisfying solved, by introducing a so-called *bias*. That is, to artificially increase the distance between the light source and the geometry. Thus, the depth value comparison is more robust, reducing the errors.

Several other extension to the shadow mapping algorithm exist. For example, Dachsbacher and Stamminger [2003] introduced Translucent Shadow Maps. For the rendering of hair, fur, and smoke Deep Shadow Maps were proposed by [Lokovic & Veach 2000].

The term *shadow map* was originally not used by Williams. The first appearance was in a follow-up paper by Reeves et al. [1987] as a figure label of the first pass depth images.

2.4.3 Hard versus Soft Shadows

The above-mentioned methods generally use point light sources. Therefore, they always generate so called *hard* shadows. That is, there is only a binary information, whether a point lies in shadow or not (see left side in Figure 2.44). *Soft* shadows, cast by area or extended light sources, produce a much more realistic image (right side).

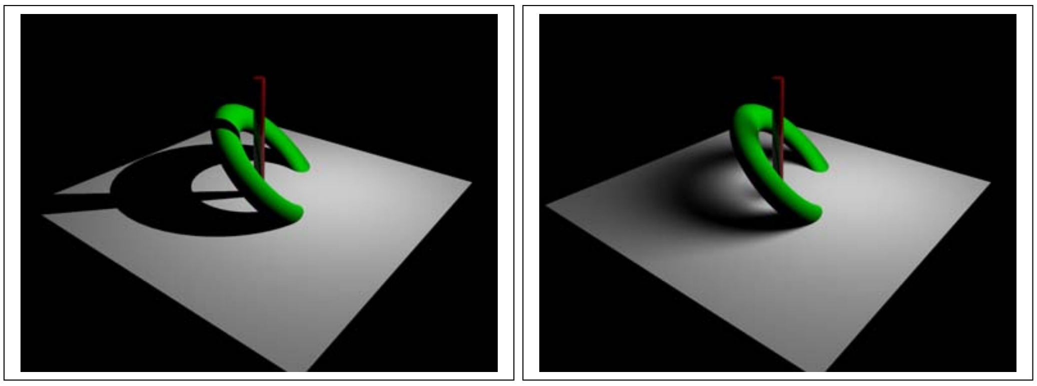


Fig. 2.44: Hard (*left*) versus soft shadows (*right*).

The soft shadow also varies dramatically with the distance from the occluder. This is not true for the hard shadows with their straight shadow boundary. This might even be mistakenly perceived as other objects in the scene.

The computational complexity to generate soft shadows lies in the fact, that not as with hard shadows, a binary information (*is the light source visible?*) for each receiver point has to be computed. Instead, the percentage of the light source seen be the receiver point is evaluated. This has to be done for all points within the penumbra region (see Figure 2.37).

There exist a lot of approximations, to carry out this evaluation [Akenine-Möller & Haines 2002; Hasenfratz *et al.* 2003], which can be divided into image and object -based approaches, whether they are based on the shadow map (2.4.2) or shadow volume (2.4.1) approach.

The following methods are based on the shadow map approach. A technique to generate soft shadow boundaries by blurring, is *percentage closer filtering* [Reeves *et al.* 1987], which also runs in hardware [Brabec & Seidel 2001].

Sampling of the area light source and the combination of a number of hard shadow images is proposed by Heckbert et al. [1997].

Soler and Sillion [1998] generate shadows for nearly parallel configurations by using convolution on occluder images.

Lazy evaluation of the visibility function is used in [Hart *et al.* 1999]. In a second phase, analytic or stochastic integration is used to compute further illumination values. This sort of filling algorithm is extend in [Jukka Arvo *et al.* 2004].

Another method, storing multiple depth samples for each shadow map pixel, was introduced by Agrawala et al. [2000].

Heidrich et al. [2000] introduce an algorithm for generating soft shadows for linear light sources, which was extended by Ying et al. [2002] to polygonal light sources.

With the help of graphics hardware, object IDs and a single shadow map, partially occluded pixels are found with the method described in [Brabec & Seidel 2002].

Wyman and Hansen [2003] introduce the *Penumbra Map* as an extension of the shadow map algorithm, Here, using object silhouette edges, a map is generated containing approximate penumbral regions.

In 2003, Chan et al. [2003] proposed to attach geometric primitives called *smoothies* to the objects' silhouettes. As an extension to the shadow map algorithm, this addition produces fake shadows, which hide objectionable aliasing artifacts that are otherwise noticeable with ordinary shadow maps.

The second class of algorithms is based on shadow volumes. The penumbra region is somehow geometrically identified and evaluated.

As with shadow maps, it is naturally possible, to combine several shadow volumes generated from samples of the light source. Due to the needed number of samples to achieve reasonable results, this method is rarely used.

A specialized approach for planar receiver surfaces was introduced in [Haines 2001]. Silhouette edges are transformed into volumes, depending on their position, simulating the effects of a spherical light source in the outer penumbra.

Another algorithm based on shadow volumes was introduced by Akenine-Möller and Assarson [2002]. Each shadow volume polygon is replaced by a penumbra wedge that encloses the penumbra region for a given silhouette edge. Then, linear interpolation of the light intensity within the found wedge results in visual shadow smoothness.

In 2003, the algorithm was generalized [Assarsson & Akenine-Möller 2003]. Wedge construction was improved and robustness increased. The real-time algorithm now handles every pixel inside a wedge.

An example for simple projective shadows is given in [Gooch *et al.* 1999]. The method was introduced with NPR for technical illustrations. Multiple occluder projection onto several receiver planes and accumulation is used to generate approximative soft shadows.

A complete other approach uses occlusion queries [Sattler *et al.* 2004b] to compute the light source visibility (see also Section 4.4.1).

2.4.4 Ambient Occlusion & Self-Shadowing

As the meaning of the word *ambient: present on all sides* suggests, ambient light has no specific origin or direction. That is, an object is virtually inclosed in a sphere and the complete inner surface of the sphere can be treated as an area light source (see Figure 2.45).

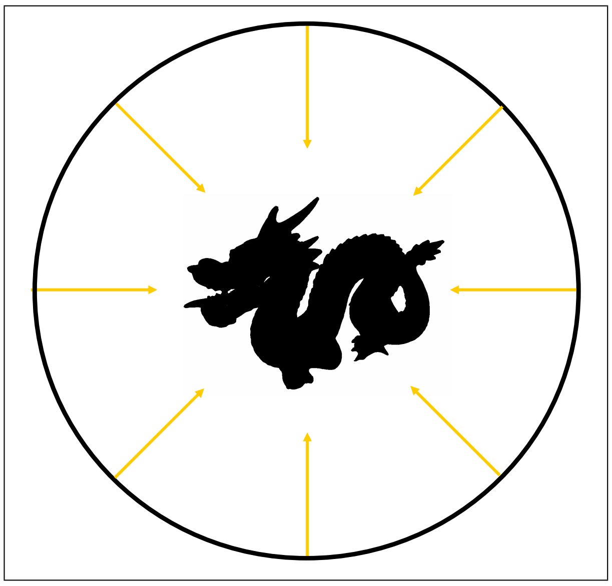


Fig. 2.45: Ambient occlusion principle.

In contrast to local shading methods like Phong shading, ambient occlusion takes global information about non-local geometry into account. On the other hand, it is a crude or average approximation to the full global illumination calculation. Only the visibility function and the cosine term are considering. The visual appearance achieved is similar to the way an object would appear on an overcast day.

Ambient occlusion is also a form of self-shadowing. That is, a part of an object is casting shadows onto other parts of itself. With this, the direct discrimination between occluder and receiver objects vanishes and is broken down to the triangle level. An example of the Stanford Dragon under ambient white light is illustrated in Figure 2.46. Self-shadowing is clearly visible for example in the mouth region.

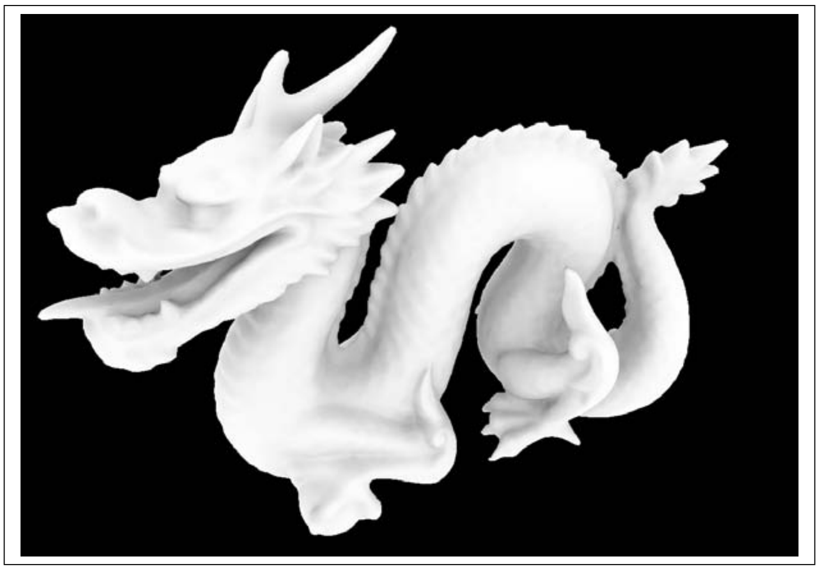


Fig. 2.46: Ambient occlusion example.

Computation of ambient occlusion is fairly complex. In contrast to the hardware-supported ambient shading, which only adds a constant color intensity to each geometric primitive, ambient occlusion evaluation requires visibility computations for every surface point, as illustrated in Figure 2.47. The point normal n defines a hemisphere above the vertex v. Red arrows denote a blocked line-of-sight to the surrounding, while green arrows contribute to the incoming light at point v. Now the percentage of green arrows to the full number of rays is calculated and a scalar value is assigned to the vertex. In the worst case of dynamic objects, this evaluation has to be done each frame.

But only ambient occlusion allows for a subtle visualization of geometric important features, like folds or wrinkles. Figure 2.48 shows a comparison between simple OpenGL shading (left image) and vertex-based ambient occlusion (right image). The superior depth impression of ambient occlusion is apparent, for ex-

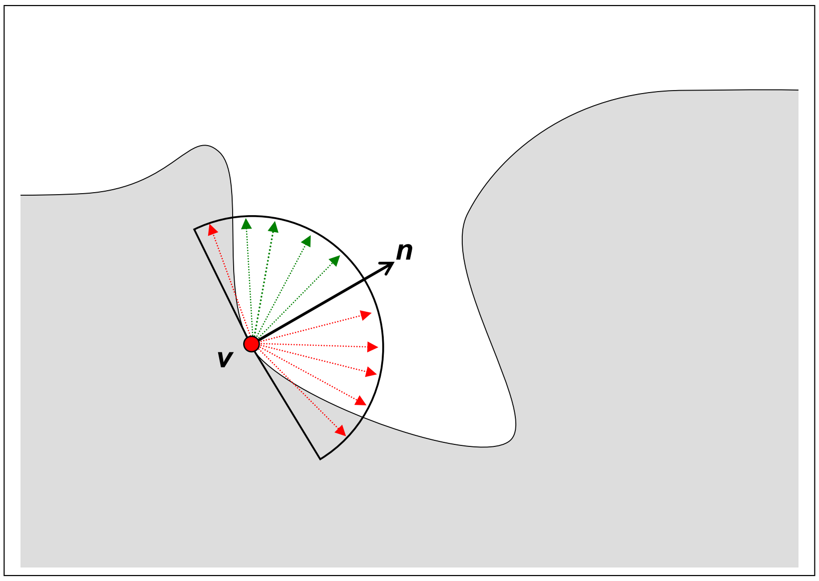


Fig. 2.47: Ambient occlusion computation for surface point *v*.

ample in the crotch area and the end of the trousers. Therefore, this techniques is also applicable to other areas such as car exterior design or terrain visualization (see Figure 2.49).

To evaluate the illumination received by a point on the surface, that is to sample its hemisphere defined by the vertex normal, there exist mainly two approaches. The first is the *inside-out* approach. That is, the evaluation is originated at the specific surface point.

As Figure 2.47 suggests the most straightforward way is to use rays to sample the hemispherical visibility around the surface of the object. This can be done either using classical rasterization or ray tracing techniques. For achieving visual pleasant results, especially ray tracing is very time consuming. The complexity of this approach depends on the number of intersection test which have to be performed and the sampling density. A speed-up might be achieved, when acceleration structures are used, but in general this approach is not very suitable for real-time applications or dynamic objects.

The second category is based on the approximation of the ambient environment by point or directional light sources, which amounts to reversing the first approach from *inside-out* to *outside-in*. That is, the visibility computation is originated at the light sources. In Practise, the model with N surface points is rendered from M

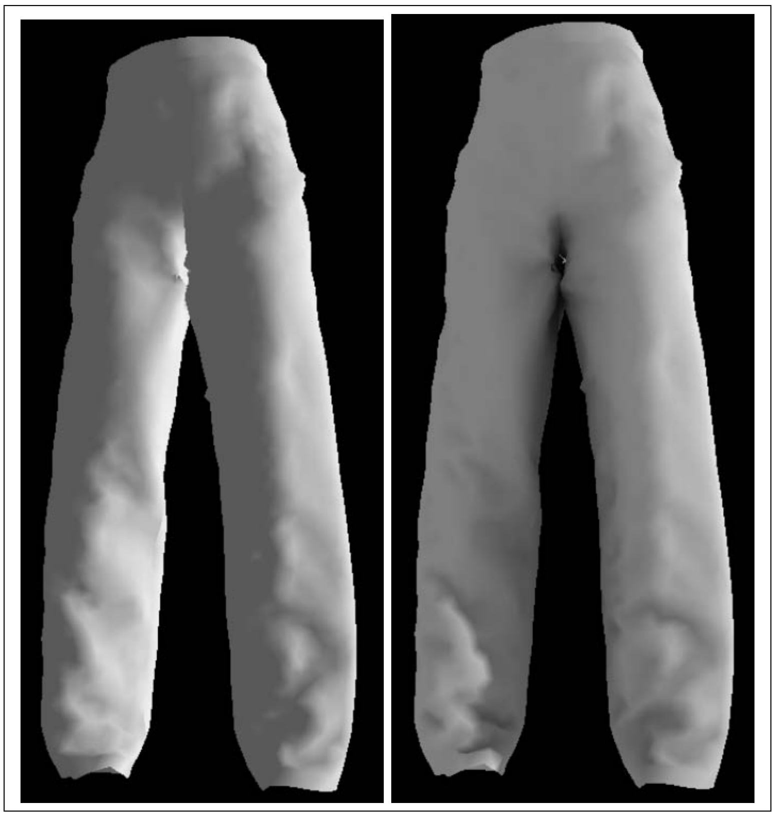


Fig. 2.48: Comparison between OpenGL shading **left** and vertex based ambient occlusion **right** for a pair of trousers.

points on a sphere surrounding the object. The distribution of these points might be random, uniform or based on some importance evaluation, for example of an environment map. With this, the complexity without optimization is $O(N * M)$. Since M is generally much smaller than N, this approach is much more interesting for dynamic objects and real-time applications, than the first one.

In the following, several methods to compute ambient occlusion are introduced.

Accessibility shading as introduced by Miller [1994] is a predecessor of the ambient occlusion method. It models the local variations of surface materials due to processes such as dirtying, aging, tearing or polishing and is capable to achieve similar visual results as ambient occlusion.

In [Zhukov *et al.* 1998] an empirical ambient illumination model is introduced. It is based on distributed pseudo light sources and local obscurance calculations,

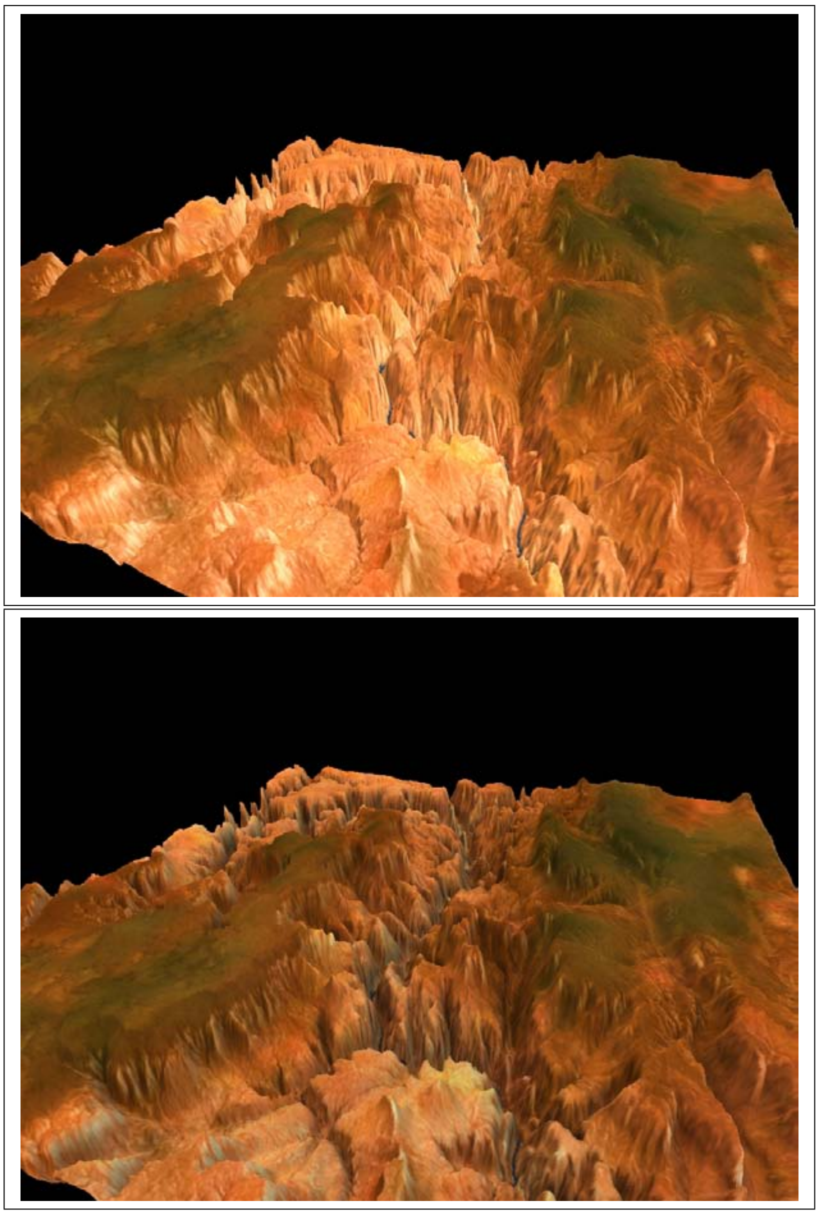


Fig. 2.49: Comparison between OpenGL shading **top** and vertex based ambient occlusion **bottom** for terrain visualization.

which are mainly distant based. Iones et al. [2003] further enhance the model.

The problem of shading folded surfaces, especially cloth, was addressed by Stewart [1999]. The main idea of his algorithm is a preprocessing step, which computes 3D visibility cones for each vertex point. The cones are stored for each vertex and used to evaluate the direct primary irradiance at runtime by doing several intersection tests. This is done by reducing the 3D visibility cones to a number of 2D visibility computations by slicing a polygonal mesh with parallel planes. Due to the geometric calculations this method is computational expensive.

Another method is to compute blocker maps [Hart *et al.* 1999] in a two step approach. First, visibility information in the image plane is computed, using lazy evaluation of the visibility function. After this, using analytic or stochastic integration, illumination values for each pixel are generated.

A Siggraph tutorial [Landis 2002] pointed out that ambient occlusion has become a popular technique in production rendering.

Visibility maps were introduced by Neulander [2003]. In contrast to ray tracing techniques, the hemispherical sampling is done via intermediate visibility maps. These maps are later combined with an arbitrary environment map to produce an approximate diffuse shaded texture.

Sattler et al. [2003] uses the standard graphics pipeline to render sides of hemicubes [Cohen & Greenberg 1985]. While this precomputation is fast, hemicubes cannot yet be evaluated efficiently on the graphics hardware, and therefore a costly read-back to the CPU is necessary.

To handle also dynamic geometry a new method based on occlusion queries was introduced in 2004 [Sattler *et al.* 2004b]. Details of this method are described in Section 4.4.1.

Related work was presented by Kautz et al. [2004]. It is mainly based on fast hemicube rasterization in order to detect blocker triangles on a per-vertex basis. For speed-up reasons, a coarser blocker mesh and a downsampled visibility mask are used. Therefore, a mesh hierarchy has to be maintained in graphics memory during run-time. On the other hand, the performance is well suited for interactive usage.

NVIDIA proposed a hardware-accelerated 2-pass method, using accumulated shadow maps [Pharr 2004; Randima 2004], which is also used in many shaders in commercial rendering software packages. To minimize sampling artifacts, jittering of the depth maps is introduced. This approach involves common shadow mapping projection problems [Kilgard 2002].

In the GPU Gems II book, Bunnell [2005] proposed a multi-pass shader, which uses disk-shaped elements per vertex to approximate ambient occlusion. The approximation is based on the solid angle of an oriented disk. This kind of LOD with the surface elements method allows for dynamic objects.

Knuth et al. [2005] used a multi layer shadow map and importance sampling of the environment map to evaluate the visibility on the GPU. Their method can also be used for non-complex dynamic scenes.

Kontkanen and Laine [2005] precompute an occlusion field in the surrounding of each object as an approximation. The volumetric information is evaluated on the GPU at run-time to compute inter-object ambient occlusion. This approach is especially suitable for computer games due to the speed and moderate storage costs.

Another method to efficiently compute ambient occlusion and self-shadowing is precomputed radiance transfer [Kautz *et al.* 2005]. Here, a linear operator that maps distant incident illumination to the neighborhood of the object, that is the full light flow from the low-frequency lighting environment is stored. Depending on the desired visual quality, precomputation times and storage costs have to be considered.

As stated in the beginning of this section, ray tracing is capable of handling globally illuminated scenes, but is naturally limited to the current camera position, and normally lacks performance for complex scenes. Interactive rates are only achieved in a massive parallel environment with optimal acceleration structures [Wald *et al.* 2003b; Wald *et al.* 2003a], which take several seconds to build. Other theoretical work [Purcell *et al.* 2002] on GPU-based raytracing is not yet available in hardware. Very recently there have also been approaches to solve radiosity on graphics hardware [Coombe *et al.* 2004], with interactive rates for small scenes.

2.5 Animation

Besides several other meanings, *to animate* is described as *to give life to* [MWD 2005]. Following this definition, animation induces a change of one or several aspects of an object or character. This might be the position, shape or color or any other perhaps more artistic aspect. In the context of computer graphics, one might say, that animation can be defined as a sequence of (still) images which are correlated in some way.

In this section, basic topics of animation are explained. With regard to methods described in chapter 3, emphasis is laid on the following aspects:

- historical overview
- animation terminology
- animation generation

Among other sources, [Watt & Watt 1991; Foley *et al.* 1996; House & Breen 2000; Parent 2002] are recommended for further reading.

2.5.1 Historical Overview

From a historical point of view, first animations, that is *a fast sequence of images* showing some kind of time-dependent change of an object, started in the 17th century with the *thaumatrope* and the *zoetrope* [Parent 2002]. Here, some mechanical image flipping was used to create the visual impression of motion.

Besides artistic animations, big steps were made in the beginning of the 20th century by the cartoonist community. Important terms here are *celluloid-based translucent layers*, *camera panning* over large drawn backgrounds or *rotoscoping* combinations of drawn images with live action.

Disney was the first, who used additional sound effects in *Steamboat Willie* (1928) to advance animation films in the direction of full-feature length films like *Fantasia* (1940).

After these early days of conventional animation the computer enters the animation field at the beginning of the 1960s. The first computer animated film was created by Edward Zajak at the Bell Labs in 1961 (*Two-Gyro Gravity-Gradient Attitude Control System*). At the same time the first video game, *Spacewar!*, was developed by Steve Russel at MIT.

The usage of computer animation in films and in the entertainment sector was and still is one of the major driving forces. Until the 1980´s computers were used for major film production and animation support. Using computers made animators life a lot easier, but most techniques used, were just adopted from conventional animation methods [Lasseter 1987].

The range for computer animation lasts from the first computer generated sequences (*Genesis Effect*) in *Star Trek: The Wrath of Khan (1982)* [STAR 1982], over liquid and morphing effects in *The Abyss (1989)* [ABYSS 1989] and *Terminator II (1992)* until the first fully computer-generated feature film *Toy Story (1995)* [TOY 1995].

Finally, with *Lord of the Rings: The Two Towers (2003)* [LOTR 2003], a fully-realized computer generated main character (*Gollum*) is introduced. For a more complete list of films see for example [CAM 2006].

Concerning computer usage, differentiation is made between *computer-assisted* and *computer-generated* animation. The first is commonly used with a human animator and here the computer provides tools, for example animation preview, for the animator to perform his task. The latter describes algorithms and methods which only use some kind of abstract description of the animation and the computer finally generates the sequence without further intervention.

2.5.2 Animation Terminology

Concerning animation the three main sections *motion dynamics, update dynamics* and *environment dynamics* are considered. Referring to an object, the first section describes the time-varying position, while the update dynamics describes properties of the object such as shape, color, transparency, structure or texture. The latter section deals with changes in lighting, camera parameters (position, focal length etc.) or the used display technique.

Most technical terms evolved with the classical 2-dimensional animation cartoons. The *storyboard* for example is a first (drawn) draft version of important points in the story (*key frames*). Then, these key frames are drawn. It is common to use some sort of *interpolation* to generate frames between key frames. Finished frames are transferred to so called *cels* (sheets of acetate film) and finally filmed. Using multiple cel layers separation between fore- and background is possible.

For creating effects like collision of objects or other physical based reactions, 2-dimensional techniques such as *squash & stretch* or *slow in & slow out* are used [Parent 2002] as interpolation schemes.

In the case of *scientific visualization*, each frame of the animation is usually the result of some numerical *simulation*. *Cloth simulation* (see also Section 2.6) is a good example for a scientific simulation, but also liquid or thermal dynamics are important topics.

Another important aspect of animation in the context of this thesis are virtual characters (humans, animals), often called *avatar* or *figurine*. A sub-topic of character animation is mimic or *facial animation*. For interaction, an avatar is a helpful psychological tool. Besides gaming and entertainment, also education, training environments, medical simulations or ergonomics studies profit from the usage of a human-like counterpart. Several properties of a virtual human are given in Table 2.6.

property	**characteristic**
appearance	· 2d-drawing, cartoon-like · 3d wireframe · body surface properties · muscles, adipose tissue · biomechanics · *cloth*, body equipment
functions	· included bones model · constraint angles · constraint forces · physical condition · physical capabilities · cognitive capabilities
individuality	· gender & age · generic character · hand-made character · racial characteristics · cultural characteristics · psychological characteristics · personality

Tab. 2.6: Overview of important animation terms.

Other aspects include time-dependent parts as *real-time interaction* or *team coordination* and autonomy parts as *communication capabilities* or *decision making*.

Hierarchical Modeling

Especially for human characters, the enforcement of connectivity of the body parts, that is the relative placement, is essential. Therefore, *hierarchical modeling* is used.

To ensure a coherent motion of all body parts, a so-called *articulated structure* or *bones model* is used. The structure contains of rigid parts (*links*) and movable connections (*joints*). A sequence of connected links without any branching is denoted by the term *kinematic chain*. The free ends of the chains are called *end effectors*. Changing the configuration of the joints is referred to as *articulation*. And a certain entity of joint angles and link positions is called *frame*.

The left most image in Figure 2.50 shows an example of a bones model, as the center images shows body parts of a figurine, which are associated to certain links and joints. The right image finally shows an example pose for different joint angles.

There exist several joint types for bones models. It is common to use two different types in computer graphics. *Rovolute* joints are joints, where one link rotates about a fixed point of the other link. The other are type are *prismatic* joints, in which one link translates relative to another [Parent 2002].

The totality of possible motion direction is described as *degree of freedom (DOF)* of a certain joint. Simple rotational joints for example therefore have only one DOF, in contrast to *complex joints*.

To represent a hierarchy its is common to use a tree structure consisting of *nodes* and *arcs* (referred to as *links* above). Connection to the global coordinate system is achieved via the *root node*. All other positions and angles are relatively orientated towards this node. Dead ends in the hierarchy are called *leaf nodes*. The latter definitions have there origin in the robotics literature.

2.5.3 Animation Data Generation

The next section describes several ways to generate animation data for computer usage. Most terms of the 2-dimensional case can be also used for 3-dimensional

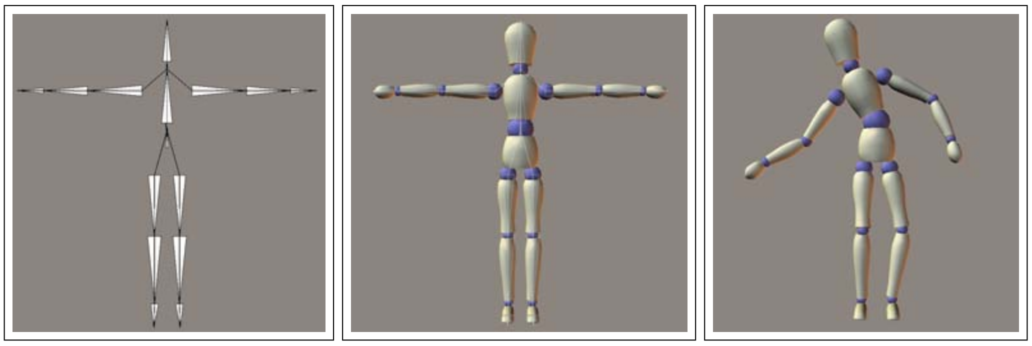


Fig. 2.50: Examples for a bones model.

animation data. In the following, always 3-dimensional computer animation data is considered.

Kinematics & Dynamics

Kinematics, as part of the classical physical mechanics, describes all parts of the motion itself, without considering the driving forces. That is, only geometric and time-dependent object properties as positions and velocities are considered. In contrast to this, *dynamics* only deals with the driving forces which cause the motion. That is, the fundamental physical laws are taken into account.

Differentiation is made between *forward* and *inverse* kinematics and dynamics. That is, in the forward case, the resulting positions, velocities and forces are calculated, while in the inverse case the necessary velocity and forces for a given end position are calculated.

Therefore, in the case of inverse kinematics, the state vector for a given position of an end-effector has to be calculated. That is, the animator defines the end position and all necessary joint angles are calculated. There exist several algorithms for calculation [Parent 2002], but no general, robust and efficient solution is known. Coming along are further inequalities, for example restrictions for bending angles.

Due to the complexity of real-world physics, sometimes so-called *dynamic constraints* are introduced, to enforce a certain behavior of an object. For example, without prior knowledge, the static friction is adjusted accordingly or reachable world positions are restricted.

Explicit Control

Using *full explicit control*, every aspect of the animation is defined. That is, every translation, rotation or scaling of any part of all objects is user-controlled. While this method naturally allows for maximal control and also allows for non-realistic motions, it is complex to handle and time consuming.

Physically Based Generation

A lot of animations deal with natural phenomena or results of simulations of the evolution of physical systems. That includes the simulation of fluids and liquids, gaseous phenomena (like wind, smoke, clouds or fire) or simple gravitational effects. Practically, the animation frames are generated out of the solutions of a numerical system, for example partial differential equations, which are often calculated off-line. Also into this section fall particle systems for modeling systems of massively present singletons, like grass blades on a meadow or swarms of birds or flies (*flocks* or *boids*).

Due to the computational complexity, often *particle-based* simulation is used. That is, instead of solving the equations for all points or properties of an object only equations of certain *key points/properties* are solved. Therefore, so-called *guide particles* are created, which influence the behavior of their surroundings. Here, a trade-off between simulation speed and accuracy is made.

The simulation of clothing (draping etc.) as an example for physically based animation generation is explained in detail in Section 2.6.

Motion Capturing

For fast generation of natural movements, *motion capturing* is used. Here, a real-world object or human (actor) performs the desired movement, while tracked via some sort of body markers. That is, instead to artificially synthesize the motion, it is measured and afterwards transferred onto a virtual character. The principle was also used back in 1915 with *rotoscoping*, that is real film sequences were used as submittal for drawings.

For the tracking, several methods and sensors exist, as shown in Table 2.7. As of today, optical infrared measurements with simultaneously captured surface textures are state-of-the-art. All methods have in common, that with all real-world measurements, noise is present and data post-processing is recommended.

system	**advantages**	**disadvantages**	**sensors**
optical	- simultaneous several objects - huge measurement area - precise - simple calibration - no harnessing	- not outdoor usable - visibility problems - labeling of markers	- body markers (passive) - body markers (active)
mechanical		- intrusive - harnessing	- multiphase motor - potentiometers - optical fibers - acceleration sensors
acoustic		- calibration - harnessing	- run time measurement - sender - receivers
magnetic	- outdoor usable	- calibration - imprecise vs. optical - harnessing	- magnetic field distortion

Tab. 2.7: Motion capture systems comparison.

One common and popular ascii-based data format is *.bvh* [BVH 2006] by the no longer existing company *BioVision*.

2.6 Cloth Visualization

In the context of virtual characters, the figurine itself, clothing and accessories (for example tools or jewelry) are important for a realistic virtual reality creation. All three parts require realistic geometry and animation for a life-like appearance.

Due to the fact, that clothing covers and decorates typically large parts of the figurine, it provides essential clues for shape, speed and motion, cultural aspects or attractiveness. Therefore, clothing contributes to a large extent to the overall appearance of a figure.

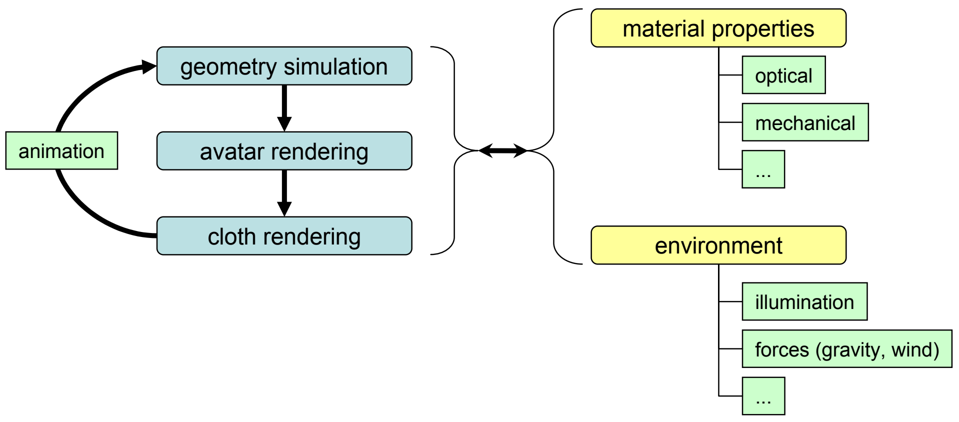


Fig. 2.51: Overview of the cloth visualization pipeline.

Figure 2.51 shows the typical cloth visualization pipeline including animation. As a cyclic process, *geometry simulation* (cloth and avatar), *avatar* and *cloth rendering* alternate. This process is influenced by external parameters and forces like *material properties* or *environmental* effects.

A well known example for realistic cloth visualization includes the Oscar-winning *Short Animated* film *Geri´s Game* by PIXAR [Pixar 2005] with calculated cloth dynamics. Or *Matrix Triology* [Matrix 2003; Borshukov 2003], in which measured optical surface properties are used for certain cloth.

Using the animation pipeline in Figure 2.51 as a basis, other aspects arise, namely avatar *motion sequence generation* and animation *data compression*, which are discussed in detail in chapter 3.2.1 and 3.3.1.

2.6.1 Historical Overview

As stated above, areas of application include virtual-try-on, animation of virtual characters for entertainment or virtual prototyping.

Requirements for the simulation include for example speed, accuracy in modeling large deformations and nonlinearities down to details like folds and wrinkles and numerical stability.

Until the mid 1980´s visualization of cloth was achieved by mapping textures to rigid surfaces only. No geometry change was calculated, instead, the cloth was modeled as a separate geometry part and hand-animated. This *static clothing* was replaced by *static draping* [Weil 1986], mass-spring-systems [Terzopoulos & Fleischer 1988; Terzopoulos *et al.* 1987] on regular grids based on Lagrange equations and elastic surface energy, and more and more complex *complete geometry simulations*, for example in [Volino *et al.* 1995; Volino *et al.* 1996].

The need for enhanced collision detection algorithms and interaction with virtual characters was introduced in [Carignan *et al.* 1992].

With the recent advance of the programmability of graphic processing units and their capacity of parallel program execution, first steps to simulate cloth entirely on the GPU have already been made [NV 2005].

The importance and practicability of cloth simulation is indicated by the integration in major graphics applications like Maya [MAYA 2006] or Poser [e-frontier 2005].

For state-of-the-art tutorials and techniques, [House & Breen 2000; Volino & Magnenat-Thalmann 2000; Hauth *et al.* 2002; Magnenat-Thalmann *et al.* 2004; Magnenat-Thalmann *et al.* 2005; Wacker *et al.* 2006] are recommended.

2.6.2 Geometry Simulation

Due to the fact, that it is rather complex to manually model deformable objects and materials, physically-based simulation is used to generate form and motion. Several techniques exist to perform the simulation, depending on the kind of objects:

- *finite element methods*
  solving a system of partial differential equations

- *particle simulation*
  space discretization in particles; solving ordinary differential equations

- *Cosserat models*
  baed on points, rods and shell; alternative to particles; engineering method

Some applications require real-time performance, like *Virtual-Reality* systems. Due to the computational complexity of the above methods, it might suffice to just produce the right *appearance* or *natural behavior* of the deformable object. That is, trading performance against physically correctness as for example in [Desbrun *et al.* 1999].

While Figure 2.51 shows several external influences and properties, cloth visualization often requires real-time performance. Therefore, certain computational complex aspects are neglected or approximated.

For up-to-date tutorials see [Hyeong-Seok *et al.* 2003] or [Hyeong-Seok *et al.* 2005].

Time Steps

Common to all methods is the discretization of space and time. Particularly important is the time discretization, that is, the simulation is split up into *simulation steps*.

Static Clothing

The computational easiest form is *complete static clothing*. That is, there is a separate geometry layer for clothing, but the complete cloth is attached to the avatar geometry and has no possibility for self-movement.

Static Draping

More realism in modeling and simulation was first introduced by Weil [Weil 1986] with *static draping*. His paper describes a method for modeling cloth material hanging in three dimensions when supported by any number of constraint points. While assuming that the cloth is rectangular weave of inelastic threads, even though this allows for the model to contain geometric folds for the first time.

Vertices can move freely on a line defined by two neighboring reference points. This line is called *catenary curve* and has the form

$$y = c - \left(a \cdot cosh \left(\frac{x - b}{a} \right) \right) \tag{2.43}$$

If the reference points are moved, the catenary curve all dependent vertices changes.

This two stage algorithm with a approximative draping within a given convex hull and afterwards a relaxation process. With this method, self-intersection of cloth is ignored.

Ignoring the physical properties of the cloth and using only geometric features naturally produces fast results [Weil 1986; Hind & McCartney 1990]. This approaches require extensive user-intervention and are not capable of reproducing realistic cloth dynamics.

Mechanical Models

Garment simulation heavily relies on the accurate usage of the mechanical properties of cloth. This properties can be measured using standardized methods [Kawabata 1980].

One method from mechanical engineering is the usage of *finite elements*. That is, the cloth surface is divided into a discrete set of patches with associated mechanical parameters, which define the shape of the patch [Collier, J.R. *et al.* 1991; Gan, L. *et al.* 1991; Eischen *et al.* 1996].

Based on energy variation, a set of equations with surface continuity constraints has to be solved for simulation.

While the mathematical formulation and tools are available from engineering finite elements fails in modeling large nonlinear deformations or highly variable collisions effectively.

Particle Systems

Particle systems for cloth simulation represent an computational easier way. Here, only a specific set of representative points (particles) are evaluated. The vertices of the cloth geometry are geometrically associated with certain particles and moved accordingly. The particles itself are moved be the mechanical forces of the cloth, by solving ordinary differential equations. Connections between particles can be modeled in several ways, including also a mass-spring system.

The main idea is the discretization of a deformable objects into a number of feature mass points (*particles*). These particles are interconnected through *springs* forming a mass-spring system. The connection topology can be adapted to the specific problem using for example different masses or spring constants. Also other constraints, for example at boundaries, can be integrated.

A particle itself is a infinitesimal small point in space, which position is described by a vector in $\mathbb{R}^3$. For animation purposes also the time-dependent change of velocity is of interest. Therefore, three positions and three impulse coordinates (6-dimensional phase space) fully describe the particle behavior. This space might be of higher dimension, if a particle systems with particle-particle interaction is evaluated. For velocity and acceleration, basic Newton´s laws of mechanics are used.

For complex simulations attention must be laid on numerical stability of the calculations. Using implicit methods can reduce calculation times to a high degree, which is crucial for certain applications, for example cloth simulation.

Among the implementation simplicity and fast computation times, particle systems are scalable and allow for complete body garment simulations [Eberhardt *et al.* 1996]. Numerical accurate models were introduced by Breen et al. [1994], which uses Kawabata data.

Complete Simulation

Terzopoulos and Fleischer [Terzopoulos & Fleischer 1988] introduce an advanced method, using a material grid consisting of springs, dashpots (acting like shock absorbers) and slip units. Therefore, deformations and stretches caused by the applied forces can be modeled. Using certain combinations of this, certain material attributes can be simulated. Figure 2.52 shows four reference vertices, which are combined with springs. The used spring constants k_i can be globally and locally adjusted.

For most realistic cloth behavior including wrinkling and bulging caused by the given avatar animation, the *complete simulation* of the cloth with the spring model is indispensable. The resolution of the cloth mesh, that is the size of the used triangles, determines the realism. Therefore, the simulation constantly has to handle

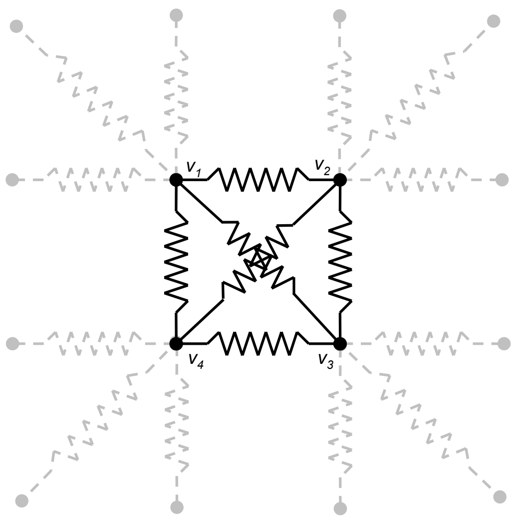


Fig. 2.52: Mass-spring model for the geometry simulation.

collision between the cloth and the avatar and cloth parts with themselves has to be detected and the response has to be computed.

To achieve a realistic motion, certain cloth characteristics such as weight, material and physical properties, even for different parts, have to be integrated into the simulation model. Examples for the properties are stretch, bend or skew abilities or wetness.

Also geometric properties as with woven fabrics have to be integrated. *Warp* and *weft* patterns of the used threads define a complex internal structure.

As stated above, collision detection and response calculation is an essential part of the simulation. Therefore, efficient algorithms have to be used to cope with certain levels of detail. To reduce the complexity organizing structures like bounding box hierarchies are used. That is, parts of the cloth mesh are subsumed into larger parts and only if global collision of the larger parts is detected, the child members are investigated.

To integrate collision response into the model, *transient springs* can be virtually attached to the figurine for example, to produce an opposing force for a cloth-figurine collision.

For realistic real-time simulation, implicit integration methods and their optimizations [Baraff & Witkin 1998; Desbrun *et al.* 1999; Choi & Ko 2002] allow fast computations for simple settings. For complex clothing, for example multi-layer garments, still heavy approximations have to be made.

Hybrid Approaches

Hybrid models combine a detailed global motion simulation with fast micro-scale approximations to generate an overall realistic appearance. Additional detail can be generated through subdivision algorithms (see also Section 2.2.1). Also other details like wrinkles can be simulated using bump- or displacement maps (see also Section 2.1.8) [Larboulette & Cani 2004].

Example-Based Methods

Data-driven techniques are also common for cloth visualization. That is, several simulations are pre-computed and learned by a system. For a given new configuration the result simulation is constructed based on the learned variants.

Examples for this approach include neural-network usage [Grzeszczuk *et al.* 1998] or *impulse response functions* [James & Fatahalian 2003].

Collision Detection & Response

If several objects are involved into a simulation, object *collision* can occur. This is also true, if boundary conditions are present, like walls or other spatial restrictions. Other fields of interest for collision detection include robotics, path-planing or haptics.

A naive approach for collision detection with n objects results in testing all possibilities with a complexity of $O(n \times n)$. This quadratic algorithm is not suitable for large n.

Therefore, this problem is efficiently broken down in computational less complex parts.

A basic idea is the two stage process of *broad* and *narrow* phase. First, using culling algorithms, all possible collisions are calculated for the current time step. That is, pairs of objects, which are too far away from each other to collide are neglected. In the second phase, costly micro-scale tests are performed, for example triangle-triangle intersection tests.

Another basic idea are *bounding volumes* (BV) or *hulls*. Here, a geometrically complex object is enclosed by a geometric primitive object like a sphere or a box. With this simplification collision tests can be done more efficiently.

There are several different volumes possible. Ranging from simple spheres, axis-aligned bound boxes (*AABB*), which reduce the problem down to one dimension, in contrast to oriented bounding boxes (*OBB*). *K-dops* are discrete orientated polyhedrons with k faces, which combine efficient intersection tests with a better object outlining.

It is also common, to form *hierarchies* out of the bounding volumes to reach another abstraction level. For instance, two human avatars are roughly described as two boxes, but in the next level of detail, each part of the body has a bounding volumes down to the fingers. That is, if the two main volumes collide, the next hierarchy level is used for collision testing and not the triangle level.

Distinction is made between *rigid body* and *deformable objects* collision. For the collision of two or n convex polyhedrons, several explicit algorithms exist [Mirtich 1998; Lin & Canny 1992]. For non-convex polyhedrons, there is the possibility to generate several convex ones and use the above methods or to use the described bounding volume methods.

In the case of deformable objects, for example clothing, two main collision types can occur. First, the deformable object can collide with the environment. In the example, cloth parts can either collide with the figurine or the ground. And secondly, the deformable object can *self-collide*, that is, parts of the object collide with other parts. Detection of self-collision can take up to 50% of the computing time [Baraff & Witkin 1998].

In the worst case scenario of an animated figurine wearing cloth, as of today the computational complexity does allow real-time results only for simplest configurations. Therefore, using approximate solutions combined with a-priori knowledge and pre-computation is state-of-the-art.

A typical example of pre-computations are *distance fields*. That is, for certain points in space, for example, in the vicinity of a figurine, the distance to the figurine is pre-calculated and transformed, if the figurine is in motion. Therefore, it is possible to compute the distance cloth-figurine relatively fast. Several distance fields also can be combined with a bounding volume hierarchy to achieve even faster detection.

If a collision is detected in the current simulation step, a *collision response* is computed. Depending on the kind of object (rigid or deformable) there exist several possible responses. Ranging from virtual time relocation to avoid collision to switching over discrete velocity changes to special object-object contact computations.

Collision detection for clothing is a field with continual research, see for example [Baraff *et al.* 2003; Teschner *et al.* 2004; Eberle *et al.* 2004; Teschner *et al.* 2005; Teschner *et al.* 2006]

2.6.3 Rendering of Cloth

Recalling Figure 2.51, the rendering of both the avatar and the simulated cloth is another mayor aspect in the visualization pipeline.

In this stage, the optical surface properties and the environmental illumination influence are decisive for the final acceptance of the resulting image. For large viewing distances, the optical appearance is much more important than the geometric accuracy.

The visually important effects like self-shadowing, occlusions, anisotropic reflection or color bleeding take all place in the so-called mesostructure layer (see also Section 2.1.19).

While for materials with nearly flat surface structures, for example *silk*, special reflection models are sufficient, structural more complex material groups, like knitwear (*wool*) require additional treatment for the shadowing aspects.

For complex real-world materials, even approaches like bump- or normal mapping (see Section 2.1.8) are not able to reproduce the *look-and-feel*. That is, information about the structure and the properties of the underlying yarn has to be created.

Modeling vs. Measuring

Two orthogonal approaches exist for creating a realistic optical model. The first is based on pure modeling of the mesostructure properties using volumetric approaches. The second is based on example-based measurement of real-world materials and the usage of advanced texturing methods.

Modeling the underlying mesostructure using volumetric textures was first demonstrated in [Gröller *et al.* 1995; Gröller *et al.* 1996]. Here, woven and knitted fabrics are analyzed and different patterns are recreated in a basic volumetric element. This pre-generated element can handle different materials and yarn thickness and is repeated according to the weaving pattern.

It can also be enhanced with different lighting models and pre-computed shadowing [Daubert *et al.* 2001; Daubert & Seidel 2002]. Rendering is either done with conventional ray-tracing or using hardware support and lookup tables.

The second approach is the direct *measurement* of all mesostructural aspects. Using this approach includes several advantages:

- real-world materials
- no complex modeling
- automatic measurement possible
- decoupling the base geometry from mesostructure

In particular the last point allows for the re-usage of measured materials on different geometries and therefore allows the transfer of optical material surface properties (see Section 2.1.4).

Because all effects are functions of viewing and illumination direction, the BTF (see Section 2.1.6) is a very efficient representation. Details of general measurement approaches can be found in Section 2.1.9.

An extensive overview tutorial, *Realistic Materials in Computer Graphics*, not only covering clothing, was held at Siggraph 2005 [Lensch *et al.* 2005]. Practical usage of the rendering aspects are shown in Chapter 6 and efficient acquisition, compression and rendering is described in detail in Chapter 5.

2.7 Data Analysis Techniques

2.7.1 Introduction

This section introduces common data analysis techniques, which are used by the methods described in this thesis. The goal of an intelligent analysis of the data is to find specific characteristics, which allow for example data compression, data reduction or efficient data reorganization. That is, these techniques are intended to find simultaneous relationships and similarities among some variables.

In the following, most techniques are applied to real-world measured data. That is, data scattering, noise or other measurement artifacts are included in the raw data.

For the practical implementation issues software packages like *LAPACK* [Lapack 2005], *newmat* [Newmat 2006] or the *Numerical Recipes* [Press *et al.* 1992] are used. Details of the following methods can be found in [Dillon & Goldstein 1984], [Jolliffe 1986] and [Berthold & Hand 2003].

2.7.2 Data Fitting and Reduction

Given some -for simplicity- 2-dimensional scattered xy-data set, *data fitting* tries to fit a mathematical equation in the form, that the distances between the equation and the data is minimized. Three classes of models are commonly in use:

- **linear:** $f(x) = \beta_0 + \beta_1 x$
- **quadratic:** $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$
- **cubic:** $f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

To obtain a simple linear regression model for a given data set, $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen in a way, that

$$\sum_{j=1}^{m} [y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)]^2 \qquad (2.44)$$

is minimal.

That is, the sum of squared differences is minimized and the obtained *least squares* estimates can be used for data prediction of unknown xy-pairs.

In the case of *data reduction* or *data compression*, the amount of data to be stored is smaller than the amount of the original data. One has to distinguish between a *lossless* and a *lossy* process. That is, it is somehow possible to reconstruct the original data without any error or not.

Most techniques to be discussed provide a simple description of the structure underlying a set of data. Generally, this can be achieved by using a few linear combinations of the original variables.

Singular Value Decomposition

Singular value decomposition or SVD is a common technique to deal with equations or matrices which are either singular or numerically close to singular. It is also a efficient technique to solve most *linear least-squares* problems (see also 2.7.4) and based on the following theorem:

Theorem 1 *Any (m × n) matrix* **X** *with $m \geq n$, can be written as the product of an (m × n) column-orthogonal matrix* **U***, an (n × n) diagonal matrix* **D** *with positive or zero elements (so called* singular values*), and the transpose of an (n × n) orthogonal matrix* **V***:*

$$X = U \quad D \quad V^T \tag{2.45}$$

For data analysis, SVD can provide information on the underlying structure of the data set. Using the rows of matrix V as vectors, these vectors are orthogonal. They also can act as unit vectors for a new re-orientated coordinate system, which is important for the following analysis, called PCA (see 2.7.3).

2.7.3 Multivariate Analysis

For multivariate analysis several techniques exist [Dillon & Goldstein 1984]. A difference is made between *dependence* and *independence* methods. The former analysis the association between two sets of variables where one set is a dependent measurement outcome. The latter analysis the mutual association between the variables, without any distinction being made between the sets.

Principal Component Analysis

In the following, a technique of the independence class named *principal component analysis* (PCA) is described (see also [Kendall 1975; Jolliffe 1986]. The primary goal of this technique is to generate a set of linear combinations of the original data variables that account as much as possible of the total variation. All extracted linear components are uncorrelated and generally account for a smaller amount of variation with higher rank. Therefore, it is easily possible to find a low dimensional representation of the data. The first c so-called principal *eigenvectors* or *principal axes* are the most informative directions and they minimize the mean square distance between the original data and the reconstructed data.

This technique is described in detail, because it is one of the main data analysis techniques, which is used by most of the algorithms described in this thesis.

In a nutshell, PCA transforms a set of n-dimensional vectors, $\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n}$ into another set of n-dimensional vectors $\mathbf{y_1}, \mathbf{y_2}, \cdots, \mathbf{y_n}$. But here, most information content is stored in the first few dimensions. Therefore, it is possible to do a (lossy) dimensional reduction by skipping the last vectors.

That is, to find a $k \ll n$ affine subspace of $\mathbb{R}^n$ such the sum of squares of the projection errors onto this affine subspace is minimized.

To practically perform the dimensionally reduction for a m-dimensional data set $\mathbf{x}$, first the sample variance (covariance) C is used:

$$C = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x_i} - \bar{\mathbf{x}})(\mathbf{x_i} - \bar{\mathbf{x}})^T \tag{2.46}$$

with

$$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i}\mathbf{x_i} \tag{2.47}$$

The *eigenvalues* of C are:

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_k \geq 0 \tag{2.48}$$

with the corresponding *eigenvectors* or *principal axes*:

$$\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3} \cdots \mathbf{e_k} \tag{2.49}$$

or

$$\lambda \mathbf{e} = C\mathbf{e} = \sum_i \langle \mathbf{x_i}, \mathbf{e} \rangle \mathbf{x_i} \tag{2.50}$$

Therefore, the eigenvectors can be written as a linear combinations of the input data multiplied with a *weighting factor* α:

$$\mathbf{e_i} = \sum_i \alpha_i \mathbf{x_i} \tag{2.51}$$

If the eigenvalues are sorted according equation 2.48 the first k eigenvectors or *principal components* correspond to the directions of largest variances that are given by the eigenvalues (times a constant).

Each original data point can be projected onto these new principal axes, but generally, only $k \ll n$ axes are used (as described above). A good criterium for the choice of k is the size of the important variance. Assuming, that all directions are equally important and using the average value for the eigenvalues:

$$\bar{\lambda} = \frac{1}{n} \sum_i \lambda_i \tag{2.52}$$

Now, one can discard all principal components which fulfil the following equation:

$$\lambda_i < \bar{\lambda} \tag{2.53}$$

Note, that PCA always assumes a somewhat linear context in the data. For nonlinear data, there exist special techniques like NLPCA (*nonlinear principal component analysis*) [Berthold & Hand 2003].

2.7.4 Clustering

Clustering is the process of identifying groups, called *clusters*, of data points in a data set. The type of the groups nor the number are usually pre-defined.

All clustering methods must cope with some questions regarding their process. First the *clustering criteria* has to be defined. That is, how the method decides to which cluster a data point is assigned and how the clusters are formed. Therefore, *similarity measures* have to be made. There exists two broad types, namely *distance-type* and *matching-type*. While the first computes some sort of distance between data with metric properties, data with qualitative properties can be handled with some matching criteria. Also, some clustering methods are recursive, that is an atomic clustering process is performed over and over again, until a *stopping criterium* is met. Generally, some sort of minimal error or the number of clusters is defined before the process starts. Figure 2.53 gives an overall overview to clustering techniques.

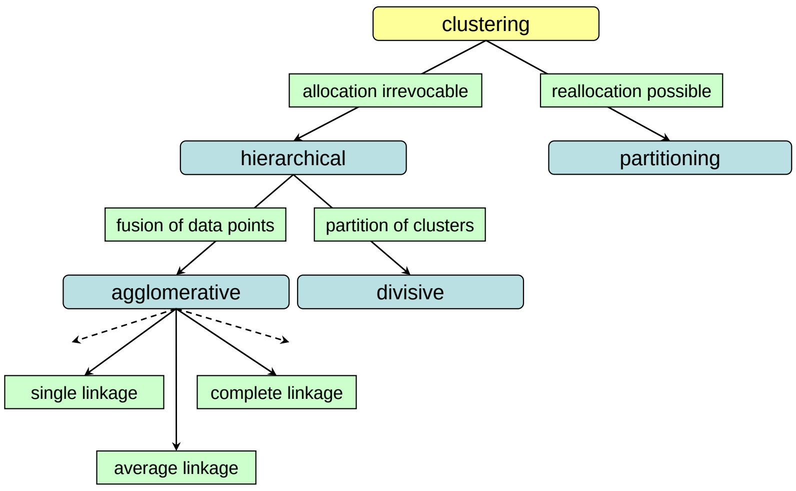


Fig. 2.53: Overview of several clustering schemes.

One primary feature of the *hierarchical* approach is that if a data point is allocated to a cluster it stays there until the end of the whole process, that is there is no way to change the cluster once assigned.

The *agglomerative method* starts out with all data points in their own cluster. That is, at the beginning the number of clusters equals the number of data points. Now, there exist several possibilities to fuse several clusters together. This is done by computing distances or similarities between clusters.

Single linkage is a nearest-neighbor method. Using a minimum-distance rule the two nearest data points are assigned to one cluster. The comparison process is repeated and either a new data point is assigned to the previous cluster or a complete new cluster is formed. In a nutshell, the distance between two clusters is given by the value of the shortest link between the clusters.

Complete linkage is a furthest-neighbor method and therefore the opposite to the single linkage. Here, the distance between two clusters is given by the value of the longest link between the clusters.

Another variation is *average linkage*. It is based on average distances, which are computed between clusters. That is, for each cluster a *cluster center* is computed and used for evaluation. Other fusion algorithms include *centroid distance* or *sum of squared deviations*.

Note, that cluster fusion is always done with the two clusters, which have the *minimum* distance to each other. And another important point to mention is that the choice of the linkage criteria leads to different clusters and must be done carefully and problem specific.

The *divisive method* is a partitioning method and therefore behaves oppositional to the agglomerative one. Here, all data points start within one huge cluster. This cluster is recursively split up into two new ones. Therefore, a *threshold distance* is defined beforehand. If the distance between the clusters is less than this threshold, the clustering process stops. This method is computational more complex and therefore only rarely used.

Both approaches show clearly that a user-defined *stopping criteria* might be necessary. In this case, when the given number of clusters is reached, the fusion or partitioning process stops.

A well known example for non-hierarchical clustering is the *k-means* method [MacQueen 1967]. With this partitioning method N data points are assigned to K disjoint subsets S_j containing N_j data points so as to minimize the following criterion:

$$E = \sum_{j=1}^{K} \sum_{n \in S_j} |x_n - c_j|^2 \tag{2.54}$$

Here, x_n is a vector representing the $n - th$ data point and c_j is the centroid of the data points in S_j [MW 2006].

The main idea with this relocation approach is to define k cluster centers or *centroids* in advance. There are several possibilities to do this which lead to different result clusters. For example, the first k data points, completely random chosen or based on prior knowledge of the data set. Next, each data point is assigned to the nearest centroid.

Now, a two-step procedure follows until the stopping criterium is met. In the first step, new geometric centroids are computed. This is done by computing the center of mass defined by all the data points belonging to the cluster. Therefore, the position of the centroid in space changes while the process evolves. In the second step, each data point once again is assigned to the nearest centroid.

With *k-means*, there is no guarantee to find the global optimum, but on the other hand it is easy to implement and usually fast to compute.

Clustered Principal Component Analysis

A straight forward combination of k-means clustering (2.7.4) and PCA (2.7.3) as introduced by [Kambhatla, N. & Leen, T.K. 1997] is *clustered principal component analysis*. Here, the PCA reconstruction error is used as distance measure.

The algorithm can be summarized as follows:

1. Initialize k cluster centers (*centroids*) r_j randomly chosen from the dataset. Assign a collection of c unity basis vectors $e_{i,j}$ to each cluster.
2. Partition the dataset into regions by assigning each data-vector to its closest center. The distance to a center r_j is given by squared reconstruction error:

$$||x - \tilde{x}_j||^2 = ||x - r_j - \sum_{i=1}^{c} \langle x - r_j, e_{i,j} \rangle e_{i,j}||^2$$

   where x is the original and $\tilde{x}_j$ the reconstructed data vector.
3. Compute new centers r_j as the mean of the data in the region j.
4. Compute a new set of basis-vectors $e_{i,j}$ per region, that is, perform a PCA in each region.
5. Iterate steps 2.-4. until the change in average reconstruction error falls below a given threshold.

CHAPTER 3 Animation

3.1 Introduction

Animate [MWD 2005]: Etymology: Middle English, from Latin *animatus*, past participle of *animare* to give life to, from *anima* breath, soul; akin to Old English *Othian* to breathe, Latin *animus* spirit, Greek *anemos* wind, Sanskrit *aniti* he breathes.

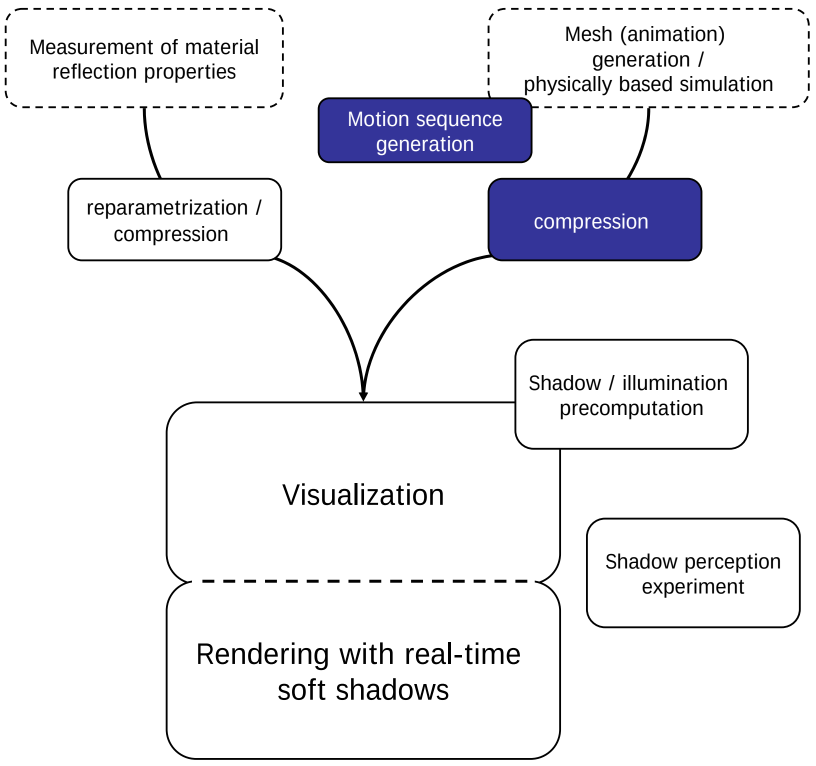

Measurement of material reflection properties
Mesh (animation) generation / physically based simulation
Motion sequence generation
reparametrization / compression
compression
Shadow / illumination precomputation
Visualization
Shadow perception experiment
Rendering with real-time soft shadows

Fig. 3.1: Overview chart with animation branch.

The main topic of this thesis is to provide methods to realistic visualize virtual cloth. Part of this is the animation of the geometry. Using the definition of *animate*, generating motion for graphical objects describes the act of animation. This chapter introduces two major aspects, which are highlighted in the animation branch of Figure 3.1.

First, a new method to generate new motions is described, focusing of the automatic generation of new animation sequences, which are based on sequences, which already exist.

After the motion sequence is calculated the transfer to the client computer for display often involves compression of the data. Therefore, efficient and simple compression schemes are needed, which are discussed in the second part of this chapter.

3.2 Sequence Generation

3.2.1 Introduction

The usage of computer generated images (CGI) and especially animations of humans or creatures in feature films [Nemo 2003; LOTR 2003; Matrix 2003] and computer games is steadily increasing. Animation sequences are also of interest in the e-commerce sector, for example for product presentations.

Creating long animation sequences with non-trivial repetitions is a time consuming and often difficult task. This is true for 2D images and more than ever for 3D sequences. Often only short basis sequences exist and have to be extended to longer sequences with non trivial repetitions (for example: a walk of an avatar). This task can not be solved by simple copy and paste techniques. On the other hand creating a large motion database or the usage of motion capture data, as it is sometimes done in the gaming industry, is also a costly task.

In this section a simple algorithm by Sattler et al. [2004b] is introduced , which is based upon the idea of video textures as introduced by Schödl et al. [2000]. It allows for the creation of new, user-controlled animation sequences, which are based only on a few key frames of the original basis sequence. This is achieved by the analysis of velocity and position coherence.

The speed and simplicity of the method is achieved by carrying out the calculations on the main principal components of the reference animation, hence reducing the dimensionality of the input data. This also leads to significant compression. The idea of video textures is extended to geometry and generalized attributes like vertex normals, velocities or reflection properties. Similar to Lee et al. [2002] control of the avatar can be achieved, but in contrast to the latter the method is not restricted to motion capture data.

The analysis of a given short basis animations leads to the creation of transition matrices, which store frame-to-frame coherence information and transition probabilities. Because the amount of data of the basis sequence might be impractical to store and to be used during runtime, compression and computation speed of the transition matrices is achieved by reducing the dimensionality of the input data, using principal component analysis (PCA).

To ensure the preservation of the motion dynamics, geometry related data (vertex positions) and animation related data (vertex velocities) is split up and analyzed independently. During runtime the algorithm creates endless or fixed length sequences, whereat the user can control the randomness and the direction of the animation. Depending on the quality of the input data two blending schemes can be used to ensure smooth animations.

Related Work

The basic idea to search for coherence in frames of image sequences was introduced by Schödl et al. [2000] with the term *video textures*. The goal of this work was to create endless or fixed length image sequences with non-trivial repetitions. For example candle flames, clocks or a waterfall animations were generated with their method. Modeling the textures as Markov processes simple frame-to-frame distances are computed and mapped to transition probabilities. Simple filtering is used to preserve dynamics and also a method for avoiding dead ends and anticipating the future is introduced.

Within this framework they also allow user control, for example a mouse-controlled fish. This was achieved by modifying the distance function and therefore the transition probability in such a way, that the valid transitions to certain frames (in the fish case towards the mouse location) are more likely than others (any other locations in the fish tank).

The latter work focuses only on image based animations, not on geometry. Geometry animation in this field naturally focuses on human motion. The sequences are driven by key frames, rule-based systems [Bruderlin & Calvert 1989; Perlin & Goldberg 1995; Bruderlin & Calvert 1996; Perlin & Goldberg 1996; Chi *et al.* 2000], control systems and dynamics [Wooten & Hodgins 1996; Laszlo *et al.* 1996; Faloutsos *et al.* 2001a; Faloutsos *et al.* 2001b] and motion capture data [Gleicher 2001; Lee *et al.* 2002].

Knowledge from biomechanics and motion studies is often involved to insure natural looking output. A lot of recent work is restricted to special animation cases, for example *diving*. Lee et al. [2002] used the video textures method to compute coherence between motion capture data to generate new data based on several input methods. They rely on a precomputed database and focus also on different control mechanisms for the avatar.

To reduce the dimensionality of the input data, Alexa et al. [2000] used PCA of the animation to compress the needed data, but are restricted to a whole given sequence. Bowden [2000] also used PCA to simplify the motion based on feature points of the objects. While Lee et al. [2002] are based on motion capture data and Bowden [2000] introduces the idea to use dimensionality reduction for the data, this section introduces the idea of combining video textures and PCA to allow the handling of arbitrary geometry animations, including geometry attributes like normals or local reflection properties.

Algorithm Overview

Motion or animation data is often given in the form of key frames containing all necessary information like vertex positions, normals, connectivity information etc. The following algorithm first does an analysis of the already finished animation A of the length l, which can be modeled by an artist, based on motion capture data, procedural created or obtained using any other form of data generation.

In the examples, animations of short or medium length ($l = 115, 190$) are used. To have a diversity in the animation, the length and hence the data amount is often much higher. But also using only a small l can involve huge data amounts. Therefore, the animation A first is compressed using standard PCA techniques [Jolliffe 1986; Kendall 1975; Press *et al.* 1992].

The motion structure is computed, using only the weights of the eigenvectors, hence can be computed efficiently depending only on the number of PCA components used.

The main idea is, that similarities between frames are transferred into the PCA weights. As in Schödl et al. [2000] coherence and transition matrices are computed, which can be used to create either infinite random or length controlled loops during rendering, as shown in figure 3.2.

3.2.2 Data Analysis

Given an animation A of l frames length and constant connectivity, the vector A_i is defined to contain all vertex positions of the geometry for frame $i \in [0, \ldots, l] \subset \mathbb{N}$. A PCA is performed on these vectors, resulting in a series of c eigenvalues λ_{ik} and eigenvectors F_{ik} $k \in [1, \ldots c] \subset \mathbb{N}$, latter will be called *eigenframes*. $\lambda_{i0} = 1$ and the mean of A are used as F_{i0}. The first $c < l$ eigenframes approximate any of the original frame A_i in such a way that the sum of the squares of the projection errors onto the affine subspace spanned by $\{F_{i0}, \ldots, F_{ic}\}$ is minimized

$$A_i \approx \sum_{k=0}^{c} w_{ik} F_{ik}, \quad i = 0 \ldots l. \tag{3.1}$$

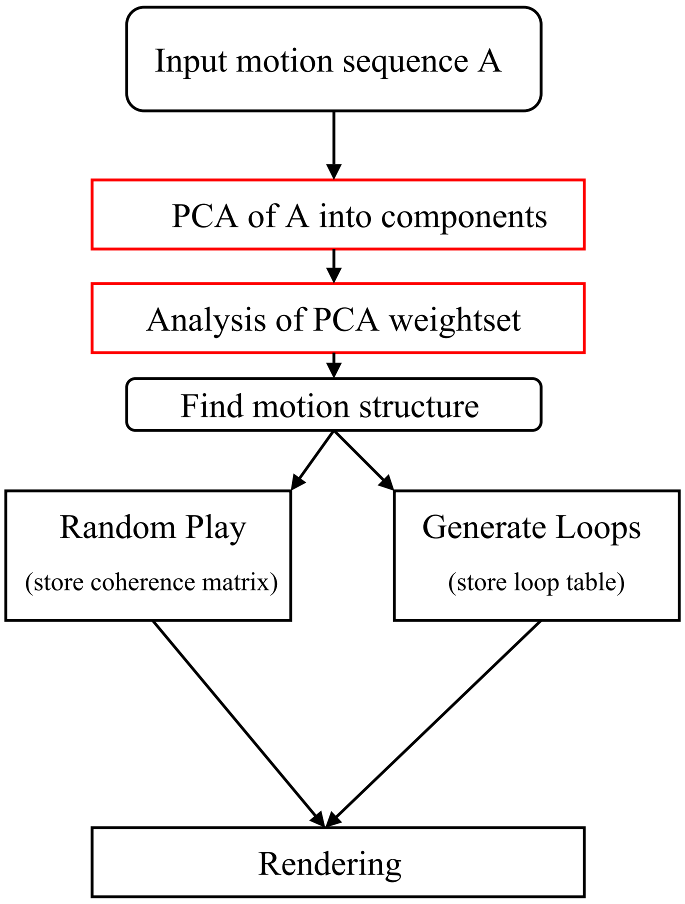


Fig. 3.2: Algorithm Overview. First a principal component analysis is used with the input motion sequence. The motion structure is evaluated using only the weight set. For random play or defined loops coherence and transitions matrices are computed, which are later used for rendering together with the eigenvectors of A.

The coefficients $w_{ik} = \langle A_i, F_{ik} \rangle$ are weights, were $\langle , \rangle$ denotes the standard scalar product in $\mathbb{R}^{3 \times N}$, where N is the number of vertices of the geometry.

Following the work of Schödl et al. [2000] and Lee et al. [2002] the data is modeled as a first-order Markov process, hence the transition between states depends only on the current state, which with this method are the given key frames in A. The Markov process is represented as a matrix P_{ij} storing the probability of a transition from frame i to frame j. This matrix is computed, by first computing the frame-to-frame distances D_{ij}, which is defined by the differences of the PCA weights w_{ik}:

$$D_{ij} = \sum_{k=0}^{c} |w_{ik} - w_{jk}| \tag{3.2}$$

As in Schödl et al., the transition probabilities from frame i to frame j are computed using an exponential function P_{ij}, as follows:

$$P_{ij} \approx \exp(-D_{i+1,j}/\sigma) \tag{3.3}$$

where σ controls the mapping between the distance measure and the probability of the transition. Higher values for σ allow for a greater variety at the cost of poorer transitions. Transitions with high probabilities are the ones, where the successor of i is similar to j, hence $D_{i+1,j}$ is small. To propagate the forward motion, the probability for frame $i + 1$ should be higher than for i itself. All probabilities are normalized per row, hence $\sum_j P_{ij} = 1$.

Motion Dynamics

To preserve the dynamics of the motion, all central velocities V_i for all vertices per frame $i, i \in [1, \ldots, l)$ are computed. Using the vertex positions p_{in} stored in A_i the velocity v_{in} for vertex n and frame i is computed:

$$v_{in} = ((p_{in} - p_{i-1,n}) + (p_{i+1,n} - p_{in}))/2 \tag{3.4}$$

where $v_{0n} = v_{ln} = 0$ is defined for all n.

Computing a separate PCA on V_i and denoting the weights as w^*_{ik}, whereas $i \in [0, \ldots, l]$ is the number of the frame the velocities are to be reconstructed for, $k \in [0, \ldots, c^*]$ with c^* the number of components used, the distance matrix is reconstructed similar to equation 3.2:

$$D^*_{ij} = \sum_{k=0}^{c^*} |w^*_{ik} - w^*_{jk}| \tag{3.5}$$

and finally

$$P^*_{ij} \approx \exp(-D^*_{i+1,j}/\sigma^*) \tag{3.6}$$

where σ^* is the mapping parameter respectively.

Another simple method to propagate the *forward* motion is to store a simple list of the last m visited frames. m should be a fairly small number ($m < 5$), to prevent certain motions, for example waving arms up and down or other jittering. While on the one hand according to the system itself this is a valid motion, it might be desired to prevent this for more natural animations.

Generate Jump Map

To create the final probability map $\hat{P}_{ij}$, which later will be called *jump map*, position and velocity coherence is used as follows:

$$\hat{P}_{ij} = P_{ij}P^*_{ij} \tag{3.7}$$

To allow different emphasis on either the position coherence (w), the velocity coherence (w^*) or both, the following extension to equation 3.7 can be used:

$$\hat{P}_{ij} = [q + (1-q)P_{ij}][q^* + (1-q^*)P^*_{ij}] \tag{3.8}$$

where $q, q^* \in [0, 1]$ control the emphasis. Note that values between $0, 1$ create a base probability for either q and/or q^* to ensure that $\hat{P}_{ij} \in [0, 1]$.
The resulting map now can be used either to generate infinite random or fixed length looped sequences.

Besides the vertex positions, the vertex normals are also included in the vectors A_i. The reconstructed normals later can be used in the rendering step for proper shading. Figure 3.3 shows examples for D_{ij},P_{ij},D^*_{ij},P^*_{ij} and $\hat{P}_{ij}$ for the used models *avatar* and *skeleton* (see also figure 3.4).

Adding vertex attributes

Besides the vertex normals, additional information could be stored with the vector A_i, for example reflections properties. For numerical stability it then might be necessary to introduce another vector R_i to store these attributes, to obtain optimal results from the principal component computations.

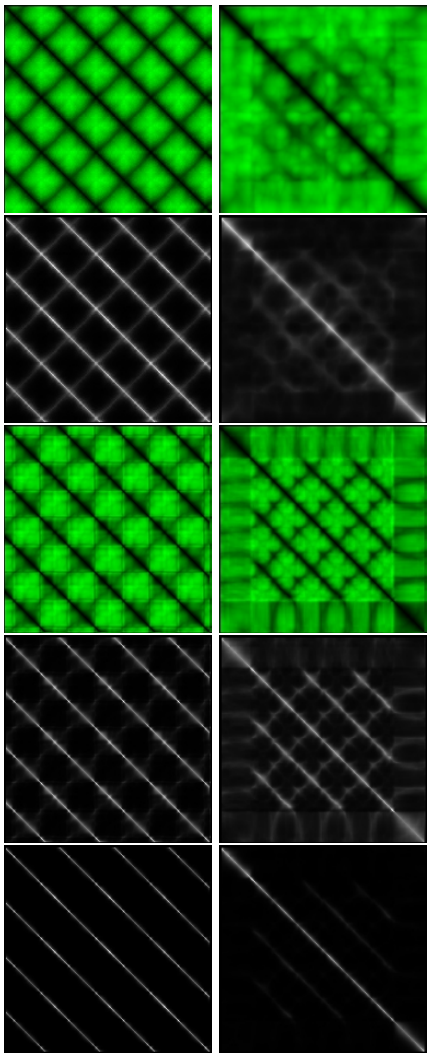


Fig. 3.3: Difference matrices and jump maps for the *avatar* (left) and *skeleton* (right) sequence. From top to bottom: D_{ij},P_{ij},D^*_{ij},P^*_{ij} and $\hat{P}_{ij}$. In the distance matrices black is zero distance and green high distance. In the probability matrices stands white for high probabilities and black for low ones. The usage of both geometry and velocity data for P^*_{ij} prevents wrong jumps.

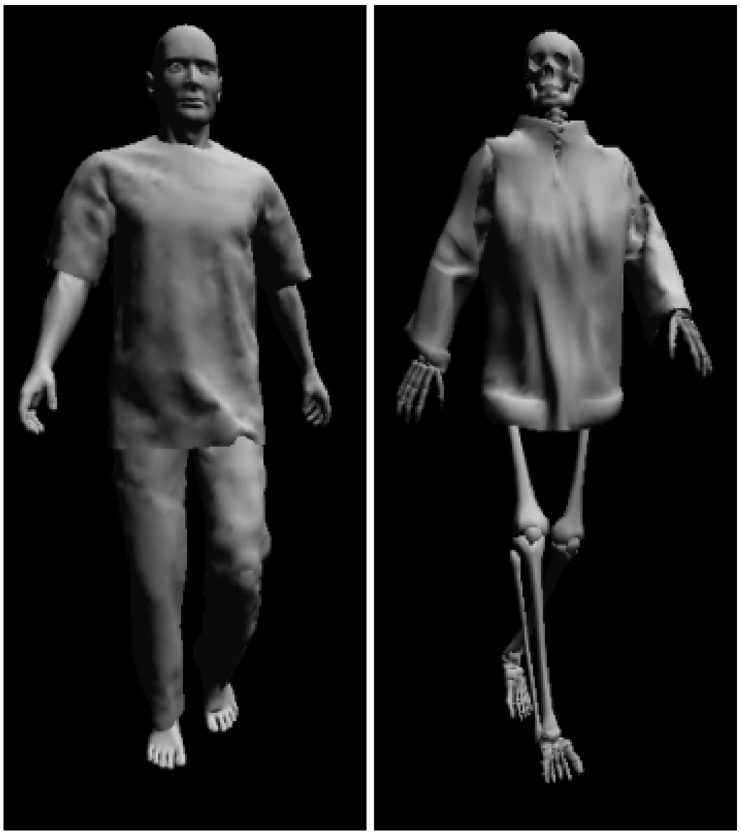


Fig. 3.4: Two models used to verify the proposed method. On the left side, the *avatar* model consists of three parts and has 20948 vertices. The *skeleton* model with another shirt on the right side consists of 12427 vertices.

Blending Transitions

Because some transitions from frame i to frame j might have large Euclidian distances, slight discontinuities can occur. Instead of jumping directly from frame i to j a chosen amount b of blending frames $i^\diamond$ is integrated:

$$i \rightarrow i_1^\diamond \rightarrow i_2^\diamond \rightarrow \ldots \rightarrow i_b^\diamond \rightarrow j \tag{3.9}$$

For the shown examples $b = 3$ yields smooth animations. The blending itself is done on the PCA weight set. Let w_{ik} be the weights for the c PCA components for frame i and w_{jk} for frame j, respectively. The blending weights $w_{i_t^\diamond k}$ for frame $i_t^\diamond, t \in [1 \ldots b]$ are linear interpolated between the weights for i and j:

$$w_{i_t^\diamond k} = w_{ik} + \frac{w_{jk} - w_{ik}}{b(b - t + 1)} \tag{3.10}$$

$\forall k, k \in [0 \ldots c]$.

Variable Length Transitions

In contrast to transitions with constant length, transitions with dynamic length can also be used. This depends on the Euclidean distance between the frames i and j. Control is achieved by introducing a smoothness parameter s, which defines the minimal distance which should be covered by a single transition step. A good starting point for s is the mean distance $s = \bar{d}$ between the frames in the animation sequence A. From this it follows that the parameter b in equation 3.9 is computed for each transition as follows:

$$b = \frac{\|\tilde{A}_i - \tilde{A}_j\|_2}{s} \tag{3.11}$$

where $\tilde{A}_i$ is the reconstructed geometry for frame i.

Motion Control

If the random or looped sequence does not satisfy the user´s needs, a simple but efficient way for more user control can be used. Instead of one input sequence, several small sequences have to be created, each containing only a specific motion *subtype*, for example *move forward*.

The user then can select a target cell (blue), as illustrated in figure 3.5 using for example a joystick. The algorithm now searches for coherence frames between the originating cell (red) and the target cell, interrupts the random jumping and continues with it, when blending to an appropriate frame within the target cell. The target cell now becomes the new originating cell. This method should work for arbitrary cell contents, as long as there is no change of the geometry connectivity.

Other control interfaces based on choice, sketching or vision as described in [Lee *et al.* 2002] could also be applied.

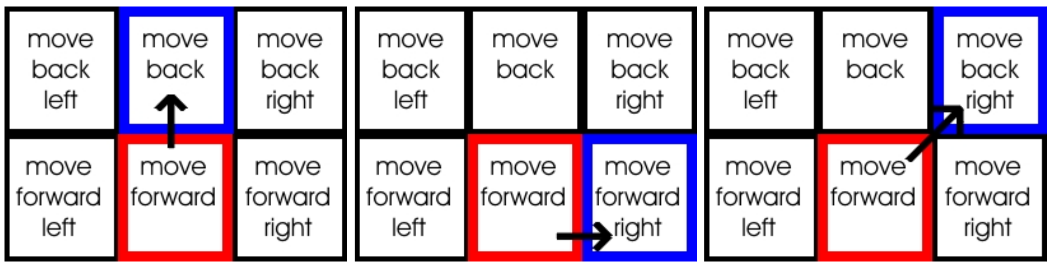


Fig. 3.5: Three examples for user control of the motion. Each cell contains PCA components for a certain motion, for example *move forward*. Only red (currently in) and blue (target) cells are valid. The blue cell is chosen through user input, for example via a joystick.

3.2.3 Results

Video II

The proposed method was implemented under Windows 2000 on a 1.5GHz Athlon with a state-of-the art graphics accelerator. All computations besides the rendering are done on the CPU. The accompanying video clips show two examples of initial animations A and two new long sequences generated. The rendering is done with OpenGL and Phong shading, using the reconstructed vertex normals.

PCA Reconstruction Error Analysis

The error of a reconstructed geometry depends on the number of PCA components used. Because the usable number might be limited through speed and/or memory restrictions two errors E and E^* are introduced, corresponding to the position and to the velocity with a given number of components c and c^*, respectively. They are defined as follows using N as the number of vertices in the geometry:

$$E(c) = \sum_{i=1}^{l-1} \frac{\sum_{n=0}^{N} \|p_{in} - \tilde{p}_{in}\|_2}{\sum_{n=0}^{N} \|p_{in}\|_2} \tag{3.12}$$

$$E^*(c) = \sum_{i=1}^{l-1} \frac{1}{N} \sum_{n=0}^{N} \frac{\|v_{in} - \tilde{v}_{in}\|_2}{\|v_{in}\|_2} \tag{3.13}$$

where E is a relative error for the geometry of the object and E^* is expressed as a relative velocity error per vertex. p_{in} is the position in world coordinates and v_{in} the central velocity of the vertex n, respectively. $\tilde{p}_{in}$ and $\tilde{v}_{in}$ are the reconstructed vertices and velocities.

Note, that this reconstruction error makes only sense to have a quality indicator for the newly generated sequence, therefore, in which amount the PCA reconstruction error determines the similarity between the original and the new sequence. These errors are discussed in section *Generated Sequences*. Figure 3.6 shows the eigenvalues for the *avatar* and *skeleton* data sets decreasing with the number of components. For the figures $c = 10$ for the geometry and $c = 5$ for velocity reconstruction were used.

PCA Compression Ratio

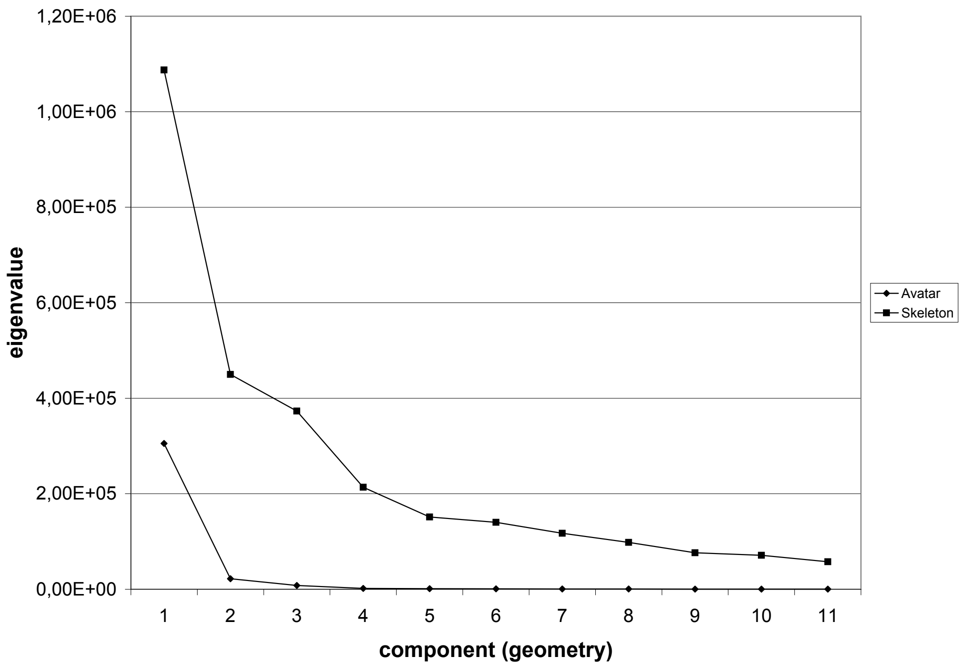
The number of used eigenvectors for the reconstruction directly effects the compression ratio of the input basis sequence. Table 3.1 shows the ratio dependent of the used number of components c for the *avatar* and *skeleton* sequence. Compare also table 3.2 for the error dependence.

c	Avatar [MB]	Ratio	Skeleton [MB]	Ratio
-	138	1:1	75,2	1:1
1	2.6	1:53	1.4	1:54
2	4.0	1:34	1.9	1:40
4	4.6	1:30	2.8	1:27
6	6.2	1:22	3.8	1:20
8	7.7	1:18	4.7	1:16
10	9.2	1:15	5.6	1:13

Tab. 3.1: Compression ratios of the used sequences for several numbers of PCA components. $c > 8$ results in visible good reconstructions, hence a compression ratio of >1:15 is achievable.

Generated Sequences

The algorithm was tested with two animation sequences, namely *avatar* (115 frames) and *skeleton* (190 frames). Figure 3.7 shows sample frames of an animation



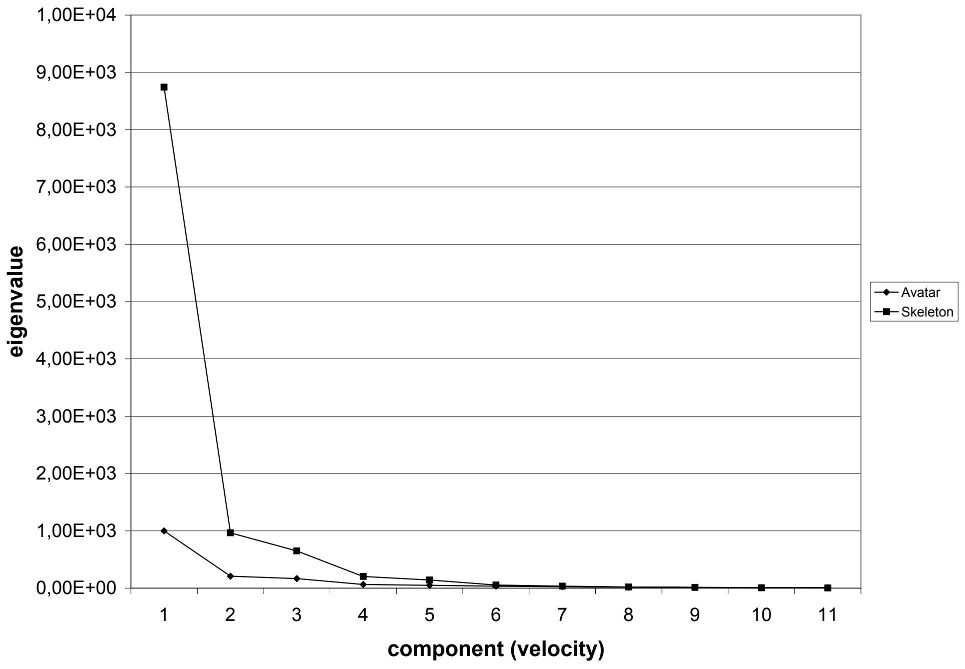


Fig. 3.6: Eigenvalues versus PCA component number for geometry (top) and velocity (bottom).

with several hundred frames. See also the accompanying video for the new animations.

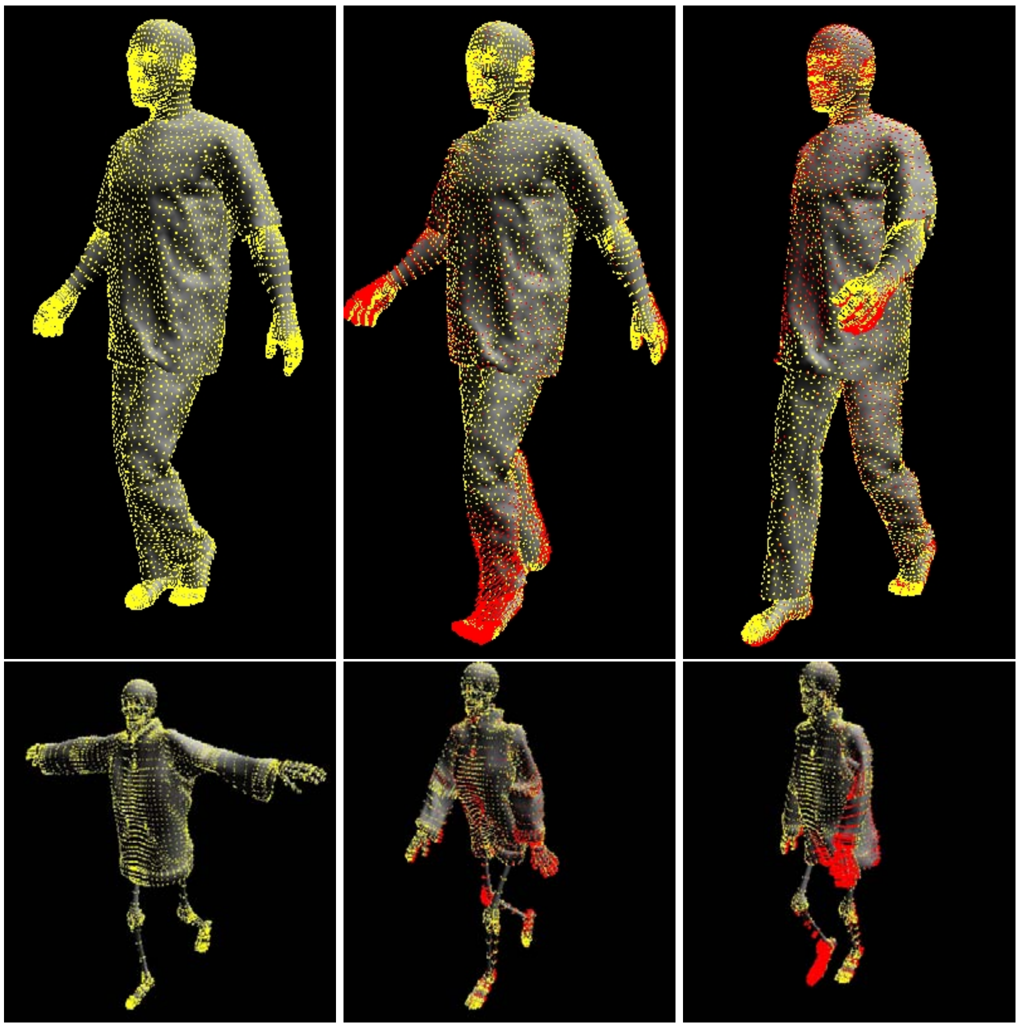


Fig. 3.7: Sample frames of the *avatar* (top) and the *skeleton* (bottom) sequence. See also the accompanying video. Yellow dots are vertex positions, red lines are velocity vectors.

Figures 3.8 and 3.9 show the influence of the parameter σ and $\sigma^*$for the two basis sequences. Normally, $\sigma = \sigma^*$ was used.

It is clearly visible, that if σ is too low or too high, instability occurs and only a specific range results in proper new animations.

For the *avatar* sequence $\sigma \approx 0.05$ generates natural animations and for the *ske-*

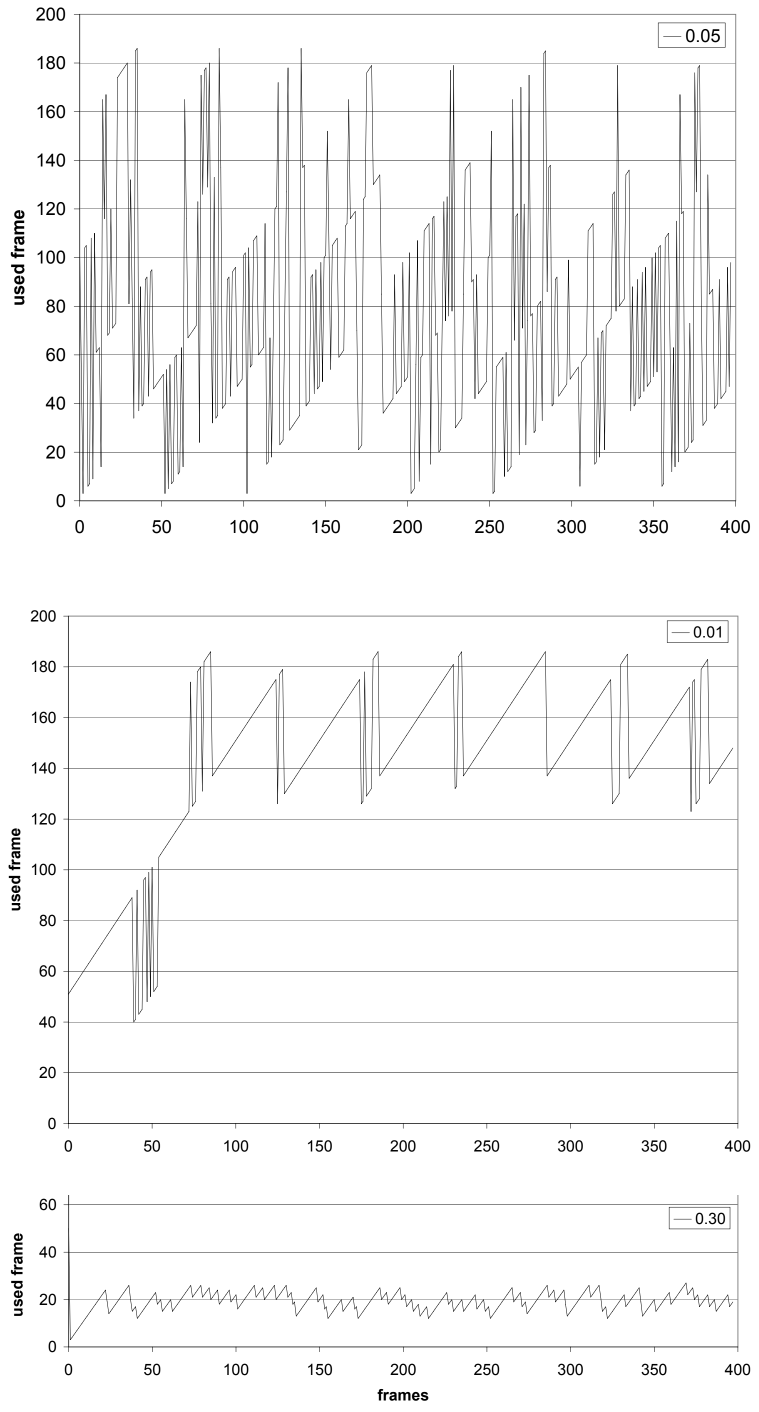


Fig. 3.8: Influence of σ and σ^* for the *avatar* sequence. From top to bottom: $\sigma = 0.05$, $\sigma = 0.01$ and $\sigma = 0.30$. Note the changing dynamics of the generated sequence.

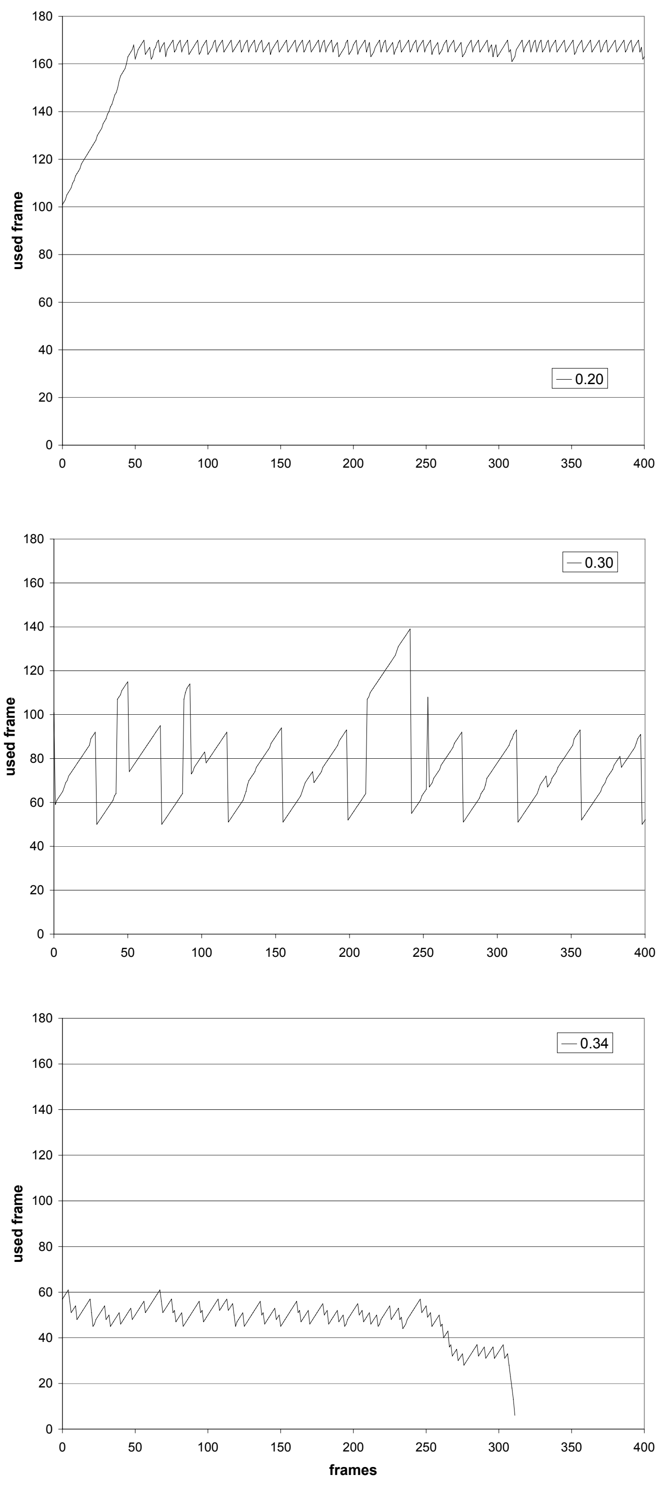


Fig. 3.9: Influence of σ and σ^* for the *skeleton* sequence.

leton $\sigma \approx 0.30$, respectively. The influence of the parameters q and q^* is shown for the *avatar* sequence in figure 3.10. If only one criterium is considered, either velocity $q = 0.0, q^* = 1.0$ or position $q = 1.0, q^* = 0.0$, the first 100 or 140 frames are sufficient to find good transitions for constant σ. If both criteria are considered, valid transitions are restricted to an extend, that the algorithm needs all frames of the input sequence or even more.

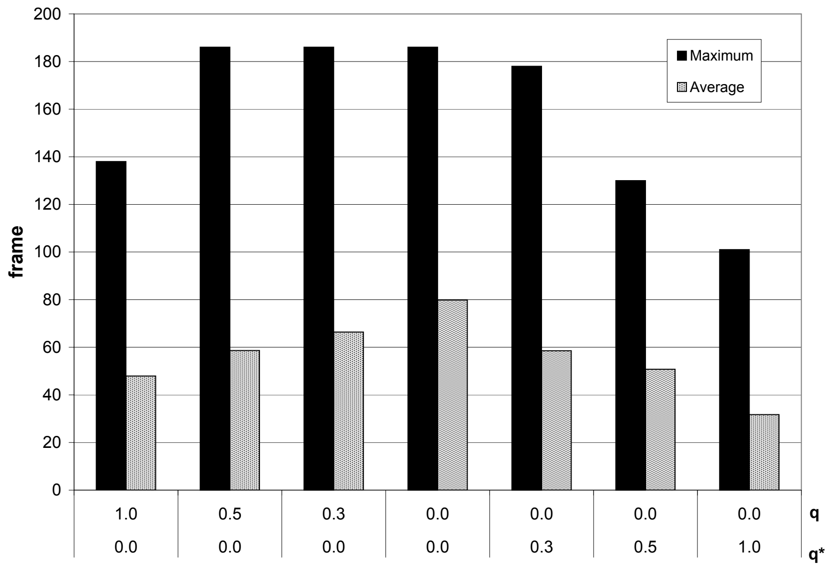


Fig. 3.10: Influence of different combinations of q and q^*, when equation 3.8 is used. See text for details.

Table 3.2 shows the reconstruction errors E and E^* for the animations depending on the number of PCA components used.

c	Avatar(E)	Avatar(E^*)	Skeleton(E)	Skeleton(E^*)
0	25.06	188.55	31.79	172.14
2	4.96	121.25	16.19	151.95
4	1.89	62.85	10.61	119.52
6	1.63	48.49	8.042	92.76
8	1.24	31.04	6.29	82.66
10	0.99	26.09	5.09	73.82

Tab. 3.2: E and E^* errors computed after equation 3.13. $c = 0$ equals to only use the mean for reconstruction. Similar to the eigenvalues the errors drop with increasing c.

Conclusions

This section introduced a method to handle arbitrary animated geometry to generate new endless or fixed length sequences with non-trivial repetitions.

The algorithm needs only a small basis animation and little preprocessing time to generate transition matrices, which are based on frame-to-frame position and velocity coherence of principal component reconstruction weights. Using the PCA also delivers a good compression of the input animation for free. Depending on the number of components used and the maximum error allowed, the complete reconstruction and evaluation of the jump map can be done at interactive to real-time frame rates on consumer hardware. This allows the easy usage of this technique in the entertainment field, for example for computer games. In practice, only the jump maps, the eigenvectors and the weights for the reconstruction besides the connectivity and texture coordinate information is needed on the client computer.

For practical usage it is worthwhile to use the increased possibilities of modern graphics adaptors. It should be possible to do the PCA reconstruction of the mesh directly on the GPU to avoid bus load. Also the random sequence generation should be possible on the GPU using texture lookups, while storing the coherence matrices as textures. To achieve even better compression results and to reduce the number of needed components local principal component analysis (LPCA), as introduced by Kambhatla et al. [1997] can be used.

3.3 Sequence Compression

3.3.1 Introduction

3D objects are often represented as polygonal meshes, which consist of vertices, edges and faces. This representation is supported by the majority of modeling tools and graphics accelerator boards are optimized to use this form of data for rendering.

For life-like soft body animations of this 3D data, for example of avatars, relative vertex positions change over time, while the face connectivity stays constant. High-quality soft-body animations are widely used in fields which allow off-line production, like films. On the other hand, this form of representation is not practical for real-time entertainment or transmission over the internet with limited bandwidth, because of the huge amount of data.

Essentially, each frame contains the whole 3D scene. Hence, efficient compression schemes are necessary. On the one hand, there should be significant compression. On the other hand, the scheme should allow for reconstruction without visible artifacts and it should be fast enough for real-time applications.

In this section a method is developed to efficiently compress the geometry data of animation sequences [Sattler *et al.* 2005b]. This is achieved by two main features. First, the data is reorganized to form vertex trajectories. Each trajectory is stored separately. After that, all trajectories are clustered using Lloyd's [1982] algorithm in combination with principal component analysis, to segment the mesh into parts which move almost independently.

These mesh parts then can be compressed more efficiently with lesser components than using standard principal component analysis on the complete animation as done in other approaches. The resulting eigenvectors and weights of the clustered PCA afterwards are compressed in the time domain. All data is further quantized to achieve an even better compression before transmission. Connectivity data is also compressed using standard techniques.

The method allows for a fast reconstruction on the GPU, using programmable shaders. Another application of the method is the user-controlled segmentation of an object, based on a sample animation.

Related Work

The storage and transmission over bandwidth-limited channels of large data is always a challenging problem. In the context of computer graphics the existing methods can be divided into the geometry compression of static and dynamic meshes.

A good overview of geometry compression is given in [Taubin & Rossignac 1999]. Most research is done to compress static geometry. Here, methods to compress geometry (vertex positions) [Deering 1995; Taubin & Rossignac 1998; Karni & Gotsman 2000], connectivity [Gumhold & Straßer 1998; Touma & Gotsman 1998; Rossignac 1999], multi-resolution [Alliez & Desbrun 2001; Karni *et al.* 2002] and progressive meshes [Hoppe 1996; Khodakovsky *et al.* 2000; Pajarola & Rossignac 2000] exist.

While it is possible to use all static compression schemes also on a frame-to-frame basis for animations, no time and space coherence can be used, which is crucial to achieve higher compression rates for animations.

To compress complete animations geometry, Lengyel [1999] proposes the decomposition of the mesh into subparts and the description of these parts as rigid-body motion. For the segmentation process only a heuristic solution is provided. Alexa et al. [2000] represent animations by using principal component analysis (PCA) to reduce the data amount. For example Nishino et al. [2001] applied PCA to the reparameterized images of an object viewed from different poses and obtained so-called *eigen-textures* and Matusik et al. [2002] compressed the pixels of captured reflectance fields applying PCA to 8 by 8 image blocks and Sloan et al. [2003a] uses CPCA for efficient radiance transfer computation.

Karni et al. [2004] use the principal component idea and also linear prediction coding to exploit temporal coherence. Briceno et al. [2003] use geometry images [Gu *et al.* 2002] to compress every frame of an animation. Reconstruction artifacts can occur due to sampling problems. A costly re-meshing process is also involved and reconstruction is only fast for small image sizes. The Dynapack algorithm as introduced by Ibarria and Rossignac [2003] exploits inter-frame coherence and uses two predictors to encode the mesh motion. Most recently, Guskov et al. [2004] uses wavelets to exploit parametric coherence in the animation sequences. While a good compression performance is achieved, a multi-resolution transform is needed and only inter-frame coherence is used. Due to the usage of wavelets, the reconstruction may become computationally costly depending on the used filter kernels.

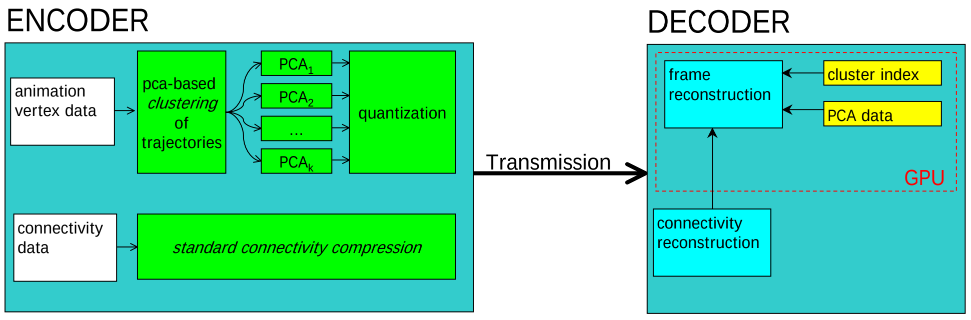


Fig. 3.11: Simplified data flow of the method.

It follows a brief overview of automatic mesh segmentation methods, because the proposed method uses segmentation to achieve better compression results. Besides the spectral compression method by Karni et al. [2000], mesh segmentation was rarely used to compress animations sequences.

For static geometry several promising approaches exist. For instance, the algorithm of Katz et al. [2003] uses geodesic distance and convexity information. Shlafman et al. [2002] propose decomposition by assigning faces to a patch based on physical and angular distances, to allow the metamorphosis between two meshes.

GPU-based mesh segmentation was shown in [Hall & Hart 2004]. Because the segmentation metric used in the following is based on the motion of a vertex present in the animation, most of the above methods are not applicable.

3.3.2 Algorithm Overview

This section describes the compression method in detail. An overview of the data flow is given in Figure 3.11. In the following a triangle mesh with V vertices and an animation with F frames in length is assumed, also constant mesh connectivity. This is true for most animations based on bones models, space warps or simulation results.

Animation Representation

Several methods exist to create 3D animations in computer graphics. Simple animations can be described with equations which model the trajectories of the vertices. But to achieve highly complex motion, the classical approaches involves human animators, who first model a character or object and then create key-frames, using techniques like inverse kinematic.

To achieve more realistic motion, physically based simulation, which describe, for instance, cloth or hair behavior are also incorporated. Based on basic animations and some constraints, new motions can be synthesized. For film and entertainment, motion capturing systems with real actors are used to animate the bones model of a virtual object. Therefore, assuming a constant face connectivity, animations can be thought of a matrix A which stores the vertex positions for each frame in its columns:

$$A = \begin{pmatrix} v_{11} & \cdots & v_{1F} \\ \cdots & \cdots & \cdots \\ v_{V1} & \cdots & v_{VF} \end{pmatrix} = \begin{pmatrix} T_1 \\ \cdots \\ T_V \end{pmatrix} \tag{3.14}$$

with V as the number of vertices, F as the number of frames in the animation and T as the vertex trajectories. The main goal is to compress this data to save memory and bandwidth.

If a lossy compression scheme is used, the reconstruction of the matrix should be done with the least error with regard to the positions of the vertices during the animation.

Mesh Segmentation and Linear Basis Decomposition

Given an animation, for example, based on physical simulation, the vertex paths through space are different and the relative relationship between the paths might show a highly nonlinear behavior. In this case, dimensionality reduction of the complete animation using PCA, which projects the sequence onto a linear subspace, will not be very efficient.

Nevertheless, many high-dimensional data sets show a local linear behavior. That is, some vertex trajectories lie similar in space, meaning that their vertices move similar within the animation. The main idea of the proposed method is to consider the vertex trajectories for compression and not the single frames, using the local linear data assumption. Therefore, instead of clustering the frames (matrix A), the trajectories (matrix A^T) are clustered. It should be the goal, to find clusters of local linear data, which is equivalent to a segmentation of the animated mesh into parts. The efficiency of the final compression relies on these clusters of the data.

As pointed out in the previous work, Lengyel [1999] proposes a heuristic solution for the segmentation, but a better segmentation can be obtained using data analysis techniques.

Therefore, a clustered PCA is applied to the animation data, which was introduced by Kambhatla and Leen [1997] to the machine-learning community in competition to classical non-linear/neural-network learning algorithms. It combines clustering and PCA using the reconstruction error as metric for choosing the best cluster.

In contrast to more sophisticated non-linear dimensionality reduction techniques, this method introduces no additional run-time cost to the reconstruction apart from a simple cluster look-up. The clustering in the encoding stage (see Figure 3.11) of the algorithm can be summarized as follows:

1. Initialize k cluster-centers r_j randomly chosen from the dataset. Assign a collection of c unity basis vectors $e_{i,j}$ to each cluster.
2. Partition the dataset into regions by assigning each data-vector to its closest center. The distance to a center r_j is given by squared reconstruction error:

   $$||x - \tilde{x}_j||^2 = ||x - r_j - \sum_{i=1}^{c} \langle x - r_j, e_{i,j} \rangle e_{i,j}||^2$$

   where x is the original and $\tilde{x}_j$ the reconstructed data vector.
3. Compute new centers r_j as the mean of the data in the region j.
4. Compute a new set of basis-vectors $e_{i,j}$ per region, i.e. perform a PCA in each region.
5. Iterate steps 2.-4. until the change in average reconstruction error falls below a given threshold.

After this process, a cluster index I_m, $m \in [1, \ldots, V]$ is generated, which stores for each original vertex trajectory T_m the number of the cluster, it belongs to.

Also, c eigenvectors per cluster are generated, which in the latter are called *Eigentrajectories*. These Eigentrajectories E_{ij}, $j \in [1, \ldots, c]$ were generated in the last step 4 of the clustering process. Furthermore, the first c Eigentrajectories approximate any of the original trajectories in their cluster i in such a way that the sum of the squares of the projection errors onto the affine subspace spanned by $\{E_{I_m 1}, \ldots, E_{I_m c}\}$ is minimized

$$T_m \approx \sum_{h=1}^{c} w_{I_m h} E_{I_m h} \tag{3.15}$$

The coefficients $w_{I_m h} = \langle T_m, E_{I_m h} \rangle$ are weights for the cluster I_m, were $\langle , \rangle$ denotes the standard scalar product in $\mathbb{R}^{3 \times F}$.

Using this method, for the given number of clusters, guarantees the best reconstruction error per cluster for a given number of components. The clustering itself heavily depends on the given model and the animation. While, on the one hand, it is possible to let the user choose the number of clusters at the onset, based on visual perception, it is also possible, to have an iterative determination of k.

The desired maximal reconstruction error or the available bandwidth for transmission can be used as a stopping criterium for increasing k, starting with $k = 1$, similar to [Katz & Tal 2003]. Clustering usually is in the order of minutes (see Tab. 3.5). Therefore, the computer can search for an optimal k.

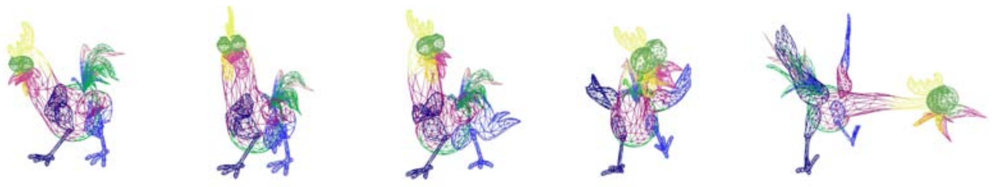


Fig. 3.12: Sample frames of the compressed *chicken* sequence with 10 clusters, each colored differently.

Results for the segmentation for some of the test animations can be seen in Figure 3.12 and 3.13. Each of the clusters is coded in a different color. Note the segmentation of the *POSER head*, which is animated by facial expressions. It shows clearly the three main parts (eye and mouth region, rest of the head).

In the case of $F \gg V$, the matrices are rearranged in a way, that the covariance matrix has the size of the smaller part of (F, V), as shown in [Navarrete & del Solar 2001]. Furthermore, due to the offline nature of this preprocessing step, out-of-core techniques implemented in packages like *LAPACK* [Lapack 2005] are used.

Error Measurement

To compare the performance of the proposed method with regard to compression and reconstruction quality, a distortion factor d_a similar to [Karni & Gotsman

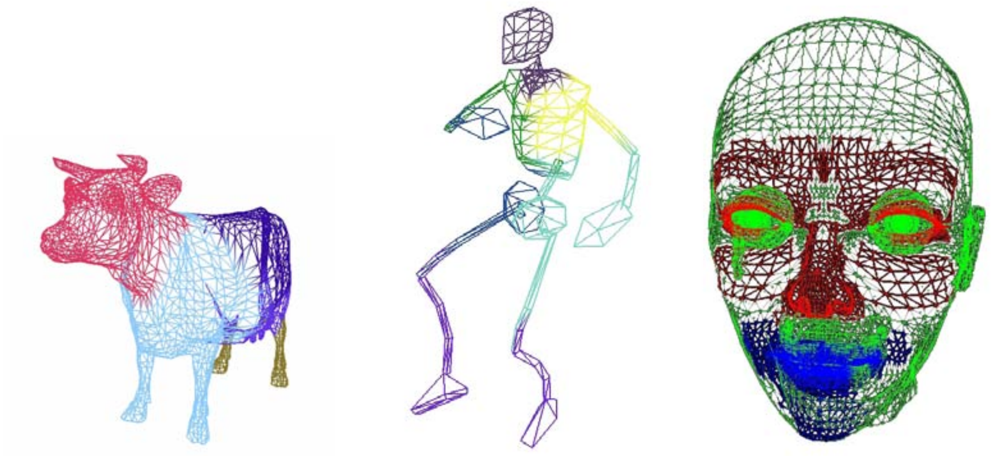


Fig. 3.13: Clustering based on animation data. Right to left: *cow* with 5, *dance* with 6 and *head* with 3 clusters, each colored differently.

2004] is introduced in the following, which measures the quality of the vertex position reconstruction for the complete animation. d_a is defined as follows:

$$d_a = 100\frac{||B - \hat{B}||}{||B - C(B)||}$$

where B is a matrix with the dimensions $3V \times F$ containing the original animation sequence. $\hat{B}$ is the same matrix after the compression and reconstruction stage. $C(B)$ is a matrix in which each column consists of the average vertex positions for all frames. Also the per-frame distortion d_f is computed, which is defined as the L2 norm of all original and reconstructed vertex positions of a frame.

Compression of Eigentrajectories

Note, that the eigentrajectories E_{ij}, which were computed above, are still of the length F. To further compress this data, again a PCA is applied to these vectors. This results in c^* new eigenvectors and the corresponding eigenvalues. This is especially useful for long sequences.

In contrast to the first step, now the time axis is compressed. After this second compression step, the new eigenvectors and weights are quantized for transmission, which is discussed in the next subsection.

Using more than $c^* = 100$ components gives nearly the same error than without time domain compression ($d_a = 0.03$). Figure 3.14 shows the influence of different numbers of components c^* on the reconstruction error d_a of the complete *chicken* animation ($k = 10, c = 20$).

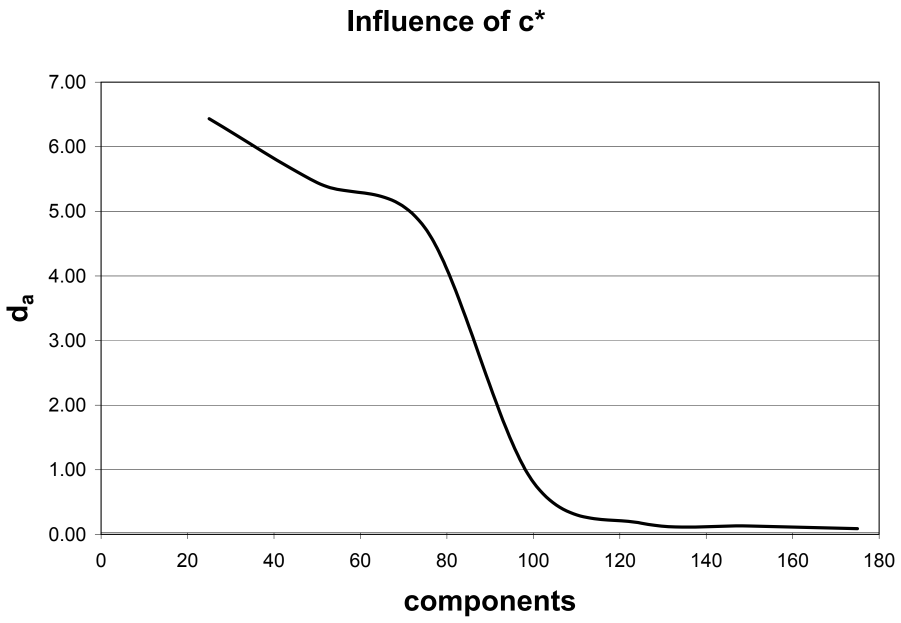


Fig. 3.14: Influence of different numbers of components c^* on the reconstruction quality d_a for the *chicken* sequence.

Figure 3.15 shows the per frame distortion d_f.

Quantization

Before storing or transmitting data, quantization is a common technique to reduce the floating-point data (32 or 64 bits), by using only q bits to represent a float value. This is done via a normalization process, which computes a tight, axis-aligned bounding box for the geometry. In the case of an animation, one first computes the center of gravity of each frame and chooses the largest occurring bounding box for all frames.

As pointed out by some authors [Karni & Gotsman 2004; Rossignac 2004] $q = 12$ bits can be treated as losslesscompression with regard to visual quality as seen in the top of Figure 3.16.

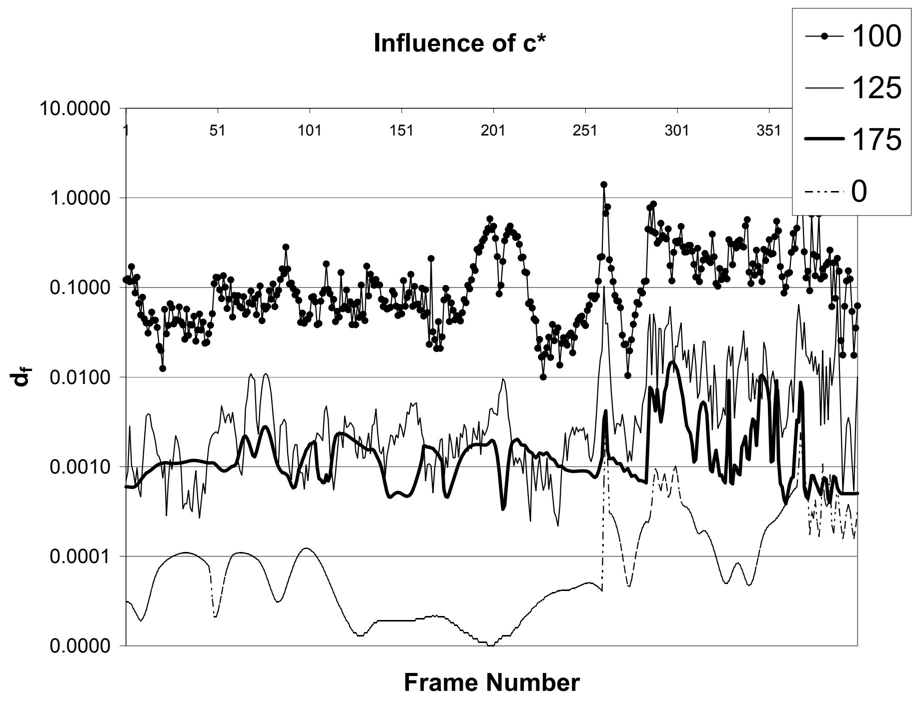


Fig. 3.15: Influence of different numbers of components c^* on the reconstruction quality per frame d_f for the *chicken* sequence.

But to operate with full precision, the PCA is computed on the original floating-point data and quantization is only performed on the components and weights as shown in the dataflow in Figure 3.11. This is done separately with two bit values q_c and q_w for the components and the weights respectively before transmitting them.

The bottom of Figure 3.16 shows the reconstruction samples for different quantization levels (top row) and different values for q_c and q_w (bottom row) for the chicken sequence with 20 clusters and 10 components.

As shown in Figure 3.17 the reconstruction error d_a is more effected by the quantization of the components, than by the quantization of the weights. In practice, the usage of at least 18 bits for both q_c and q_w is advised .

Decompression

After transmission, at first, the Eigentrajectories are decompressed. After this, k clusters, each with c PCA components and the corresponding weights from the

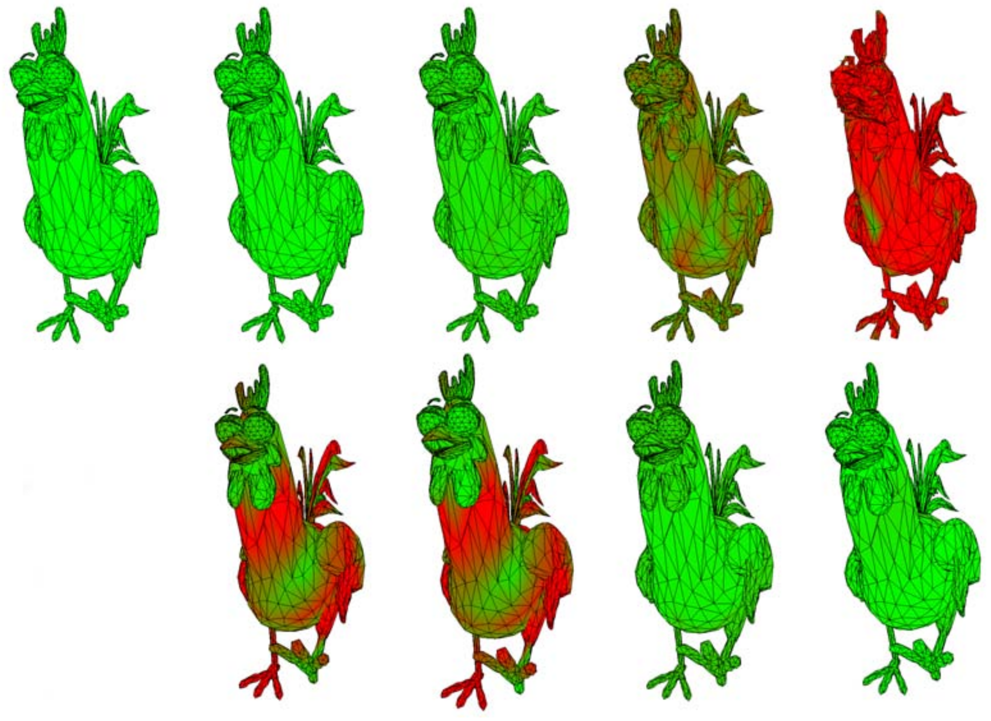


Fig. 3.16: Top row: Different quantization levels (q=32,12,10,8,6 bits) of the original data for the *chicken* sequence . **Bottom row:** Reconstruction results using different quantization for q_c and q_w (q_c=12,q_w=12/q_c=12,q_w=32/q_c=32,q_w=12/q_c=16,q_w=16). Color-coded error to original 32bit frame. See text for details.

clustering process are restored. The reconstruction of each trajectory of the animation, hence each frame, is now possible. In addition, the cluster index I_m is also restored.

As shown in Figure 3.11 the reconstruction can easily be done on the GPU, because all data can be stored in graphics memory and only simple matrix multiplications have to be computed to reconstruct a certain frame. This can easily be done within a shader program [OSS 2005].

Figure 3.18 gives an overview of the GPU shader program written in GLSL pseudocode for decompression. The cluster index and the weights are stored in 2D textures, while the PCA data is stored in 3D Textures. After several lookups the new vertex position is calculated.

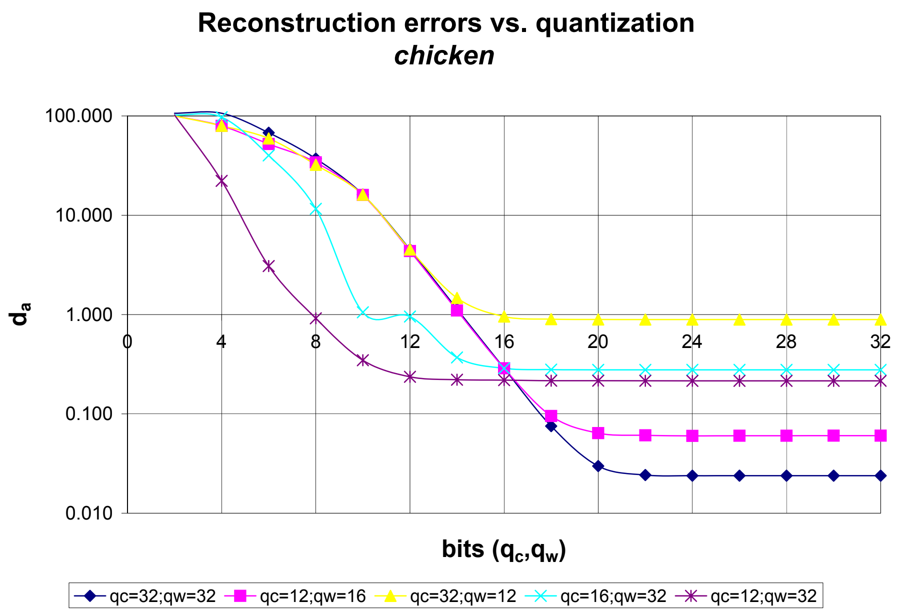


Fig. 3.17: Reconstruction errors for different combinations of q_c and q_w for the *chicken* sequence.

```
for all vertices $j$ do
    vec4 newVertexPos;
    floatclusterIdx = texture2D(texClusterIdx, texCoord[j]);
    vec4 weights = texture2D(texWeights, texCoord[clusterIdx]);
    newVertexPos.xyz = texture3D( texClusterMean, texcoord[clusterIdx]);
    for $i = 1$ to $NumComponents$ do
        newVertexPos +=
        weights[i]*texture3D(texClusterComponents, texcoord[clusterIdx]);
    end for
    glPosition = glModelViewProjectionMatrix * newVertexPos;
end for
```

Fig. 3.18: Pseudocode to compute new vertex positions in a shader program for decompression

3.3.3 Results

The following section describes the data used and introduces some error measurements, followed by an evaluation of the method and the comparison to other compression schemes.

Data Sets

All of the following data sets are animations of polygonal meshes and the number of faces and their connectivity does not change over time. The animations were done either by hand, by the usage of motion capture data or pre-generated basic facial expressions. Table 3.3 shows basic information for all animations sequences.

name	vertices V	triangles T	frames F
chicken	3030	5664	400
cow	2904	5804	204
dance	452	570	1733
dolphin	6179	12337	101
face	539	1042	10001
head	8172	15974	500

Tab. 3.3: Used animation sequences.

Some of the animations can be viewed in the accompanying videos and sample frames can be seen in Figure 3.19.

Video III

Video IV

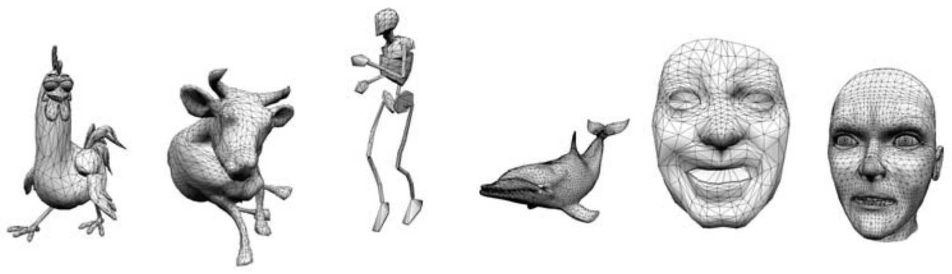


Fig. 3.19: Sample frames of the used animations. From left to right: *chicken*, *cow*, *dance*, *dolphin*, *face* and *head*.

Compression & Reconstruction Results

For the evaluation of the method described here, the reconstruction error metrics described in the subsection *Error Measurement* and the *bpvf* (bits per vertex per frame) unit for bandwidth usage measurements are used.

name	**bpvf**	d_a	c^*	c	k
chicken	8.7	0.002		60	2
	4.7	0.076		20	10
	2.8	0.139		20	5
cow	7.4	0.16		40	5
	3.8	0.50		20	5
	2.0	1.47		10	5
dolphin	7.1	0.024		20	2
	4.1	0.033		10	4
	2.1	0.168		10	2
head	7.4	0.003		40	10
	5.3	0.003		60	2
	2.5	0.083		20	5
dance	6.1	0.21	-	5	10
	4.9	0.25	40	5	10
	4.3	0.48	35	5	10
	2.5	0.87	-	2	10
	1.9	2.06	15	2	10
	1.3	4.57	10	2	10
face	10.0	0.013	-	5	20
	5.1	0.023	50	5	20
	5.0	0.029	-	5	10
	2.5	0.038	25	5	10
	1.5	0.058	15	5	10

Tab. 3.4: Compression results.

The face connectivity is transmitted beforehand as *payload* and the data is distributed over the whole animation for comparison reasons. Note, that the *Edgebreaker* method with freely available source code by [Rossignac 1999] is used to compress the connectivity information. In the worst case $6V$ bits to store the data are needed. Table 3.4 summarize the compression results for different bpvf for the used animations. Note, that c^* is used only for the long sequences.

Timings

The compression was done using an AMD Athlon64 XP 3200+ with Windows 2000. As graphics adapter an ATI Radeon X800pro was used. Table 3.5 shows compression and decompression timings for several test animations. Compression times are in seconds for the clustering, principal component analysis and sa-

name	**clusters** k	**components** c	t (sec)	FPS
chicken	2	60	258	105
chicken	10	20	206	214
chicken	5	20	395	215
cow	5	40	75	145
cow	5	20	59	218
cow	5	10	55	284
face	10	40	2730	648
face	2	40	979	654
face	5	20	1320	865

Tab. 3.5: Compression/Decompression timings for several animations.

ving. Frames per second (FPS) for display while reconstructing. It seems, that the performance heavily depends on the number of clusters used.

Comparison

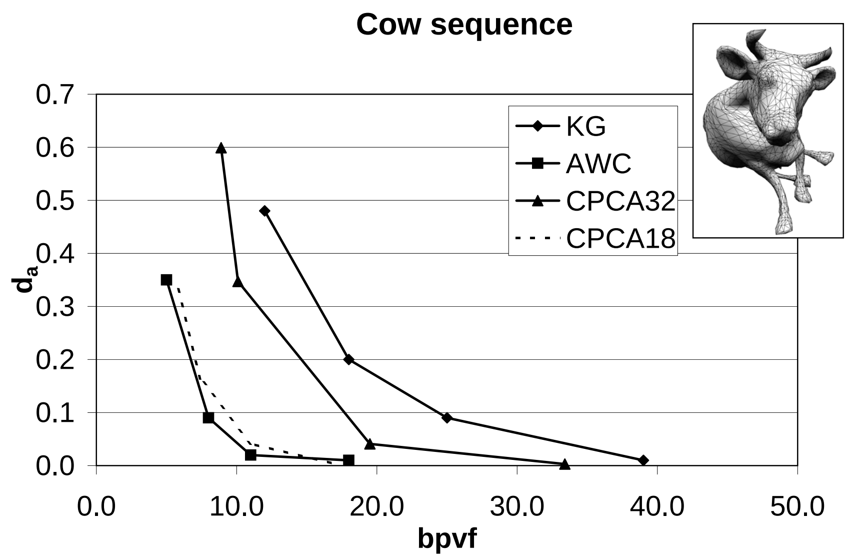
In comparison to other approaches the new method performs very well. Comparison is done against the wavelet (AWC) [Guskov & Khodakovsky 2004] and linear prediction coding (KG) [Karni & Gotsman 2004]. The values were obtained from the corresponding papers. Figure 3.20 shows a graphical result for the *cow* sequence.

Depending on the quantization level used (32 or 18 bits for the PCA components), results come close or even outperform all other methods. The right side of Figure 3.20 uses fixed bpvf to visualize d_f. k=5, c=40; k=10, c=20 and k=5, c=20 are used for d_f=7.4, 5.7 and 3.8 respectively. Besides this, the method outperforms the KG method for long sequences, as can be seen in Figure 3.21.

To calculate the bpv per frame (bpvf) the following equation is used:

$$bpvf = \frac{6V + q_c kc3F + q_w cV + 5V}{FV} \tag{3.16}$$

with V and F as the number of vertices and frames of the animation. The first part of the equation encodes the connectivity, the second part the PCA data (with k as the number of clusters and c as the number of components), the third part the PCA weights and the last part represents the cluster index encoded with 5 bits. The cluster index can be avoided, by rearranging the data in a way, that all trajectories are ordered by the cluster number. Then, equation 3.16 becomes:



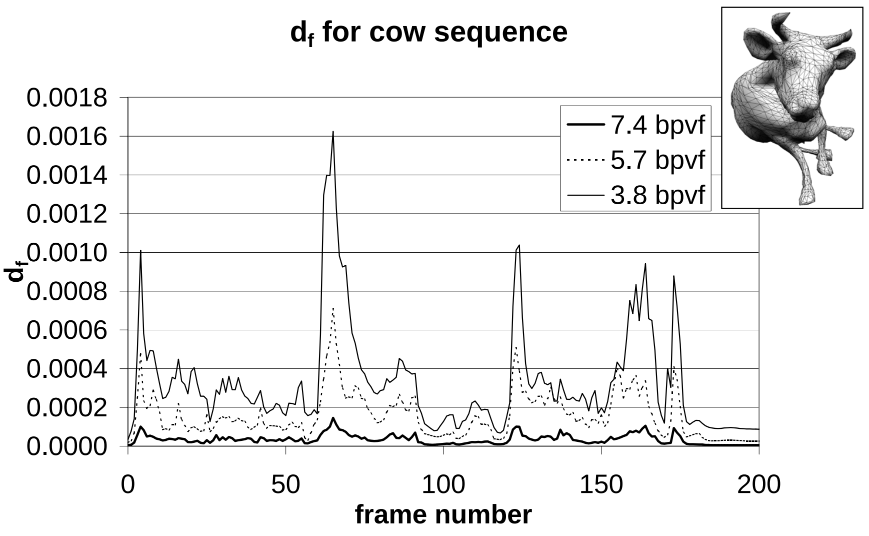


Fig. 3.20: . Left: *Cow* sequence comparison to other methods. The CPCA method with 18 bit PCA quantization yields very good results. Left: d_f for the cow sequence with different bpvf settings.

$$bpvf = \frac{6V + q_c kc3F + q_w cV}{FV} \tag{3.17}$$

using some *stopbits* to indicate the cluster change. Note, that the latter will not allow for random access the trajectories, which might become to costly during de-

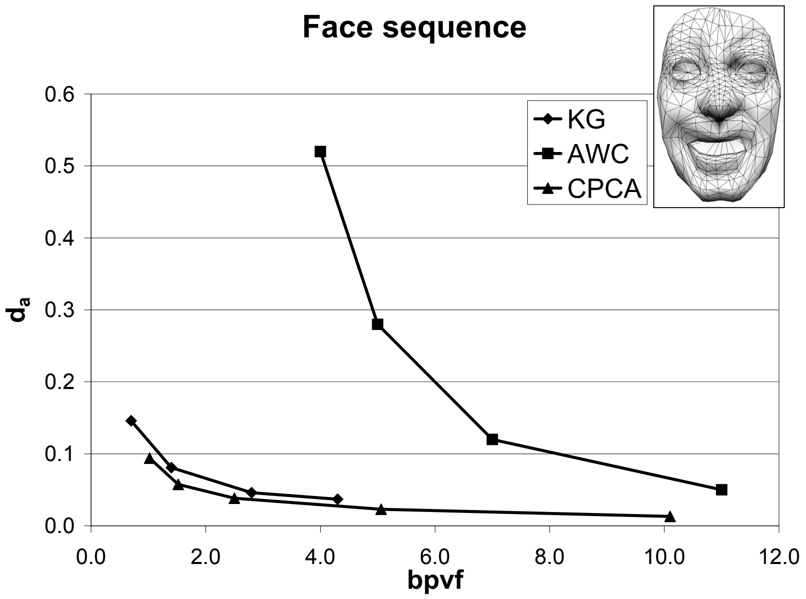


Fig. 3.21: . Comparison of the proposed method with other algorithms for the *face* sequence.

compression. With compression of the Eigentrajectories for long sequences 3.17 becomes:

$$bpvf^* = \frac{6V + q_c kcc^* + q_w 3Fc^* + q_w cV}{FV} \tag{3.18}$$

using $c^* < F$.

Discussion and Conclusions

To achieve efficient compression of the 3D data of a soft-body animation sequence, the exploitation of the spatial and time correlation of is crucial.

The method uses clustered principal component analysis to separate the given geometry into coherent parts, in regard to the animation. The animation is not treated as a series of static meshes and frames, instead it is rearranged to analysis the trajectory of each vertex individually. Compression is then done on the trajectories of all vertices of a cluster. Compression rates of the sub-meshes outperform simple principal component compression or combination with linear prediction encoding and for some animation types even wavelet-based methods. If the number of frames is much higher than the number of vertices in the animation, further

PCA-based compression in the time domain of the Eigentrajectories is performed.

The method requires no meta knowledge besides the geometry data, for example no bone model information. Given sufficient animation data, it allows for automatic segmentation of the mesh. Memory problems which might can occur during the principal component analysis, can be handled using out-of-core methods. Reconstruction is easily done on the GPU.

The method can be combined with linear prediction coding, which might lead to even higher compression rates. As shown in the result section, using a different number of clusters in combination with the distortion error, the method might be able to predictä meaningful number of parts for the segmentation of the object. In this version of the algorithm the number of components per cluster is fixed. Given a global distortion error, the number of components per cluster might individually be changed and a cluster-wise different quantization on a local basis can be applied.

CHAPTER 4 Shadows

4.1 Introduction

Shadow [MWD 2005]: Etymology: Middle English *shadwe*, from Old English *sceaduw-*, *sceadu* shade
1 : partial darkness or obscurity within a part of space from which rays from a source of light are cut off by an interposed opaque body
2 : a reflected image
3 : shelter from danger or observation

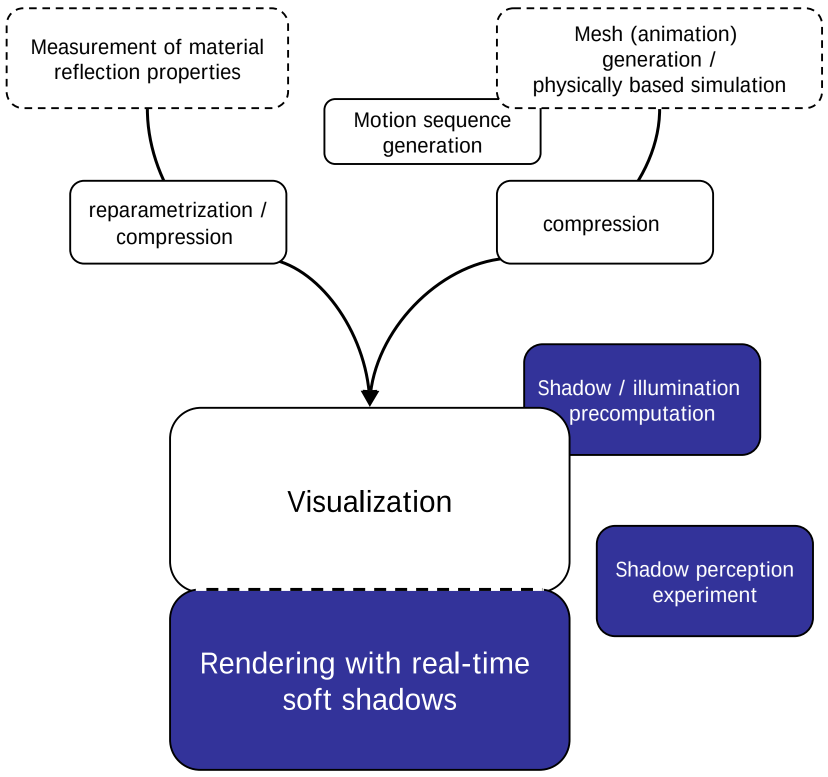

Measurement of material reflection properties
Mesh (animation) generation / physically based simulation
Motion sequence generation
reparametrization / compression
compression
Shadow / illumination precomputation
Visualization
Shadow perception experiment
Rendering with real-time soft shadows

Fig. 4.1: Overview chart with shadowing parts.

Realistic cloth visualization without correct shadowing, including self-shadowing, is unthinkable. Within this chapter mayor aspects of shadowing are tackled, as highlighted in Figure 4.1.

Starting with an experiment to evaluate the human shadow perception, approaches are described, that allow for the speedup of rendering, due to the fact that the average human observer is insensitive to slight errors or approximations in the shadow rendering.

For the visualization of macroscopic geometry details, such as folds and wrinkles and the correct handling of ambient lighting, two algorithms for correct self-shadowing are introduced in the following.

While the first is integrated into the BTF rendering pipeline and is suitable for pre-calculations, the more advanced algorithm uses hardware-accelerated visibility calculations and is capable of interactive to real-time frame rates.

4.2 Perception

4.2.1 Introduction

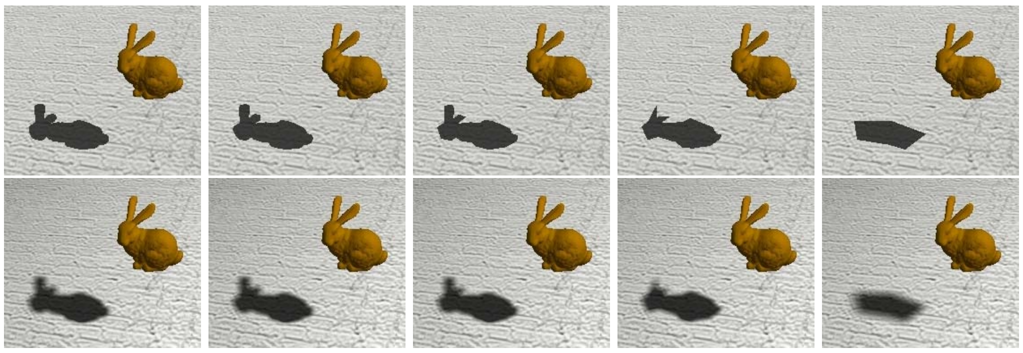


Fig. 4.2: Visual perception of shadows. Decreasing level-of-detail for the shadow caster object from left to right. Hard shadows cast by a point light source (top row) and soft shadows cast by an area light source (bottom row) are shown.

Shadows are an important visual clue about the spatial structure of an object. Using virtual reality applications for example, such as life-sized cloth visualization, or medical surgery planning, they are important because they increase the presence of the virtual objects and the overall realism. For the entertainment industry, where realistic images and high frame-rates are desired, this is also an important aspect.

However, the shadows need to be computed at interactive frame rates, otherwise usability and presence will break down. While local illumination models are the strength of modern graphics hardware, more advanced techniques which are capable of generating for example soft shadows are notoriously hard to perform efficiently. For the overall realism of a computer generated image, the used illumination is also important. While the classical graphics adapter pipeline only supports artificial point light sources, realistic images would require area light sources or image based illumination with environment maps.

But, if the indispensable ingredients to generate a realistic image are shadows cast by area light sources, to what extend have these shadows to be correct, to be visually accepted by a human observer? If it is true, that the human observer is not very good in detecting slight errors in shadow visualization, it would be possible to use a simplified model of the object to cast the shadows, instead of the original one and achieve the same *acceptance rates* of the resulting images.

Therefore, the main idea of the following experiment by Sattler et al. [2005a] is to evaluate which level-of-detail (LOD) is sufficient for the shadow casting object to produce acceptable shadows.

4.2.2 Related Work

There exists a lot of work on shadow perception. Wanger et al. [1992; 1992] have done experiments about the context of object spatial position and size and shadow shape and sharpness for simple objects. Other experiments show that shadows are an import visual clue for object-object contact [Hu *et al.* 2000; Madison *et al.* 2001]. The importance of object motion and shadows for spatial perception was investigated by Kersten et al. [1996; 1997].

However, a vast amount of work has also been published on rendering shadows. Shadow maps [Williams 1978] or shadow volumes [Crow 1977] and all derivates are the classical approaches for point-like light sources. Both algorithms are well suited for todays graphics adaptors [Kilgard 2002]. An important detail of the shadow volume algorithm is the detection of silhouette edges. A silhouette edge is an edge, where one of the normals of the corresponding faces points towards the viewer (or the light source) and the other points away. Therefore it is necessary to test all edges, each time the light source moves. There exist some approximations [Markosian *et al.* 1997], which might lead to artifacts.

Recently, more advanced techniques for soft shadow generation were developed, using various techniques, like wedges [Akenine-Möller & Assarsson 2002; Assarsson & Akenine-Möller 2003; Assarsson *et al.* 2003], smoothies [Chan & Durand 2003] or penumbra maps [Wyman & Hansen 2003]. A good overview of algorithms which produce soft shadows can be found in Hasenfratz et al. [2003].

Mesh simplification is one of the fundamental techniques used for polygonal meshes, there is an extensive amount of different methods. Details are explained in Section 2.2.1.

4.2.3 Experimental setup

The program used for the experiment (see Figure 4.3) shows two different images of the same scene. The left side of the screen shows a high resolution version of the shadow casting object above a plane. This high resolution version is also used to calculate the shadows. The object shown on the right side is also the high resolution version, but in contrast to the left side, the object to calculate the shadows is a simplified version of the original.

In the bottom area of the graphical user interface several buttons for program control can be seen.

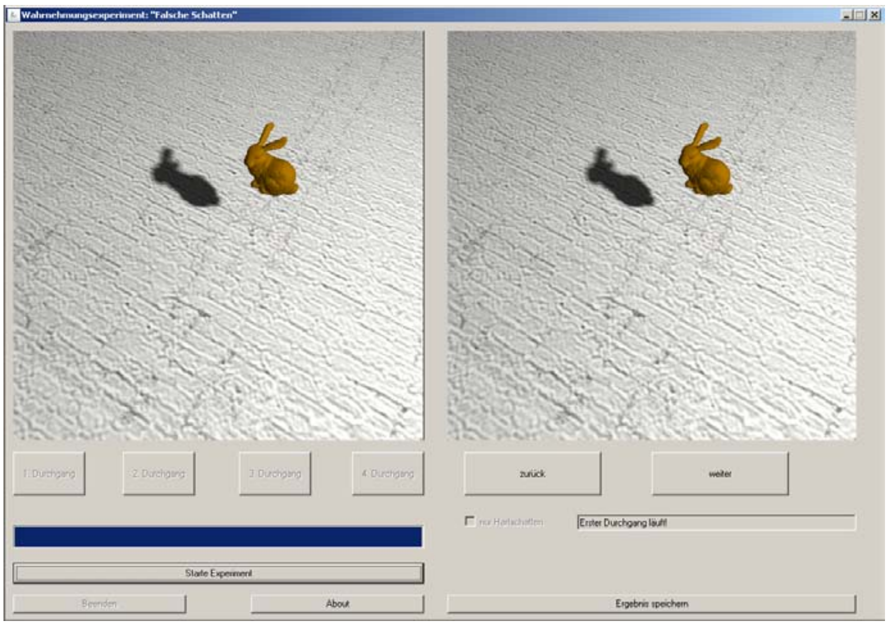


Fig. 4.3: Program used for the perception experiment

To compute the shadows, a modified version of the soft shadow algorithm [Assarsson *et al.* 2003] is used. This version uses shadow volumes, to compute the umbra region. As test object the *Stanford Bunny* [Curless & Levoy 1996] consisting out of about 70000 triangles is used. The simplified versions of the shadow casting object were generated using techniques from [Borodin *et al.* 2003]. Thirty six level of detail were precalculated, ranging from 100 (LOD=1) up to 50000 triangles (LOD=36). In each level the number of triangles is increased. See Figure 4.4 for details.

In the current version of the experiment, the shadow receiver is a plane. As the distance between the object and the plane twice the object size is chosen, since this allows a wide range of viewing angles from which both object and shadows are fully visible. Increasing the distance further would make side views impossible, while reducing the distance would increase the minimum viewing angle above 30 degrees. Note, that the radial size of the area light source also is chosen accordingly. During the experiment, the test person is able to move the light source and the point of view around the object, while the viewing distance is fixed. Thus, it is possible to examine the generated shadows under several viewing angles. The current LOD can also be changed interactively. On the left side of the screen, always the highest LOD with shadows is shown for comparison.

The Hausdorff-Error between each LOD level and the original mesh is precalculated. This allows for the comparison between the original mesh and the simplified version. In Figure 4.4 the Hausdorff-Error is given in percentage of the bounding box diagonal.

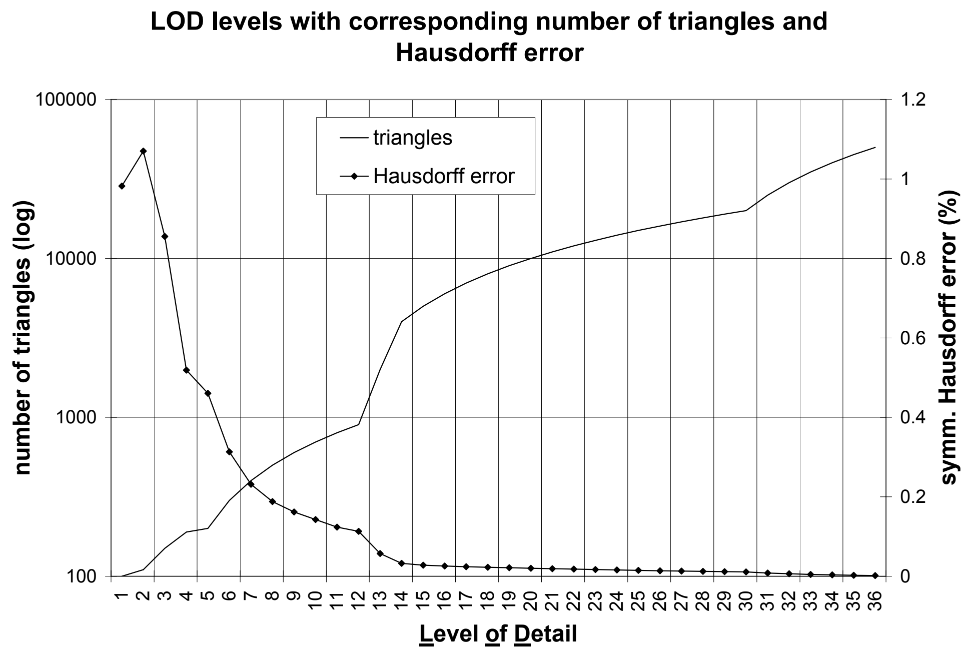


Fig. 4.4: LOD numbers with corresponding triangle count and Hausdorff error.

4.2.4 Experimental procedure

The experiment is performed in the following way. Perception is tested in two directions. First, starting with the highest level-of-detail of the mesh (LOD=36), the LOD number decreases. That is, the number of triangles used for calculating the shadows, decreases. Following the tradition of the *Just Noticeable Difference* experiment by Weber [1834], the test person can mark the level, which seems to be the first to produce noticeable errors in the shadows. The second part of the experiment starts with the lowest LOD level and the test person this time marks the level, which seems to produce correct shadows for the first time.

These two parts are repeated with a different size of the light source. This allows predictions about the influence of the size of the penumbra on the shadow perception. Then, the area light source is substituted by a point light source and the experiment is performed again, now with hard shadows. For the experiment, 20 test persons were interviewed.

4.2.5 Results

Because the perception of shadows by an observer differs from human to human, there can not be an exact transition point between *realistic* and *artificial* perception, but more or less a transition region, in which most observers will fall. The results of the first two parts of the experiment (area light sources) are shown in Figure 4.5. In the first part of the experiment 17 test persons (=85%) are between 200 (LOD=5) and 600 (LOD=9) triangles. 8 (=40%) of these persons were satisfied with a shadow cast from only 200 triangles and 25% from 300 triangles.

In the second part of the experiment (increasing LODs) nearly the same results were observed. The mean value for the large area light sources is around LOD=5.6. The third and fourth part of the experiment (larger light source) show similar results as the first and second part, as shown in Figure 4.6. It seems, that the size of the penumbra region has only little influence on the perception. The mean value for the large area light sources is around LOD=5.4.

To summarize the results for soft shadows and the object *Stanford Bunny*: less than 1% of the original number of triangles for shadow calculation are sufficient to produce realistic shadows for a majority (=90 %) of the test persons. On the computer which was used for the experiment, the rendering time per frame for the original mesh is about 1.15 seconds. Using only LOD=9, a speed-up of factor 16 can be achieved (0.07 seconds rendering time). In the last two parts of the experiments *hard shadows* are rendered.

The scattering of the hits is more obvious, as in the parts with soft shadows, as can been seen in Figure 4.7. For both directions, the majority of test persons (=87.5 %) is between LOD=5 and LOD=17. Besides that, some outliers can be seen at LOD=30 and LOD=36. The mean value for the point light source is around LOD=10.9. It seems, that errors in hard shadows are noticed more early, than in soft shadows, which is reasonable, since soft shadows blur fine details. The possible performance gain in rendering speed is about factor 7 from 0.129 seconds for the original

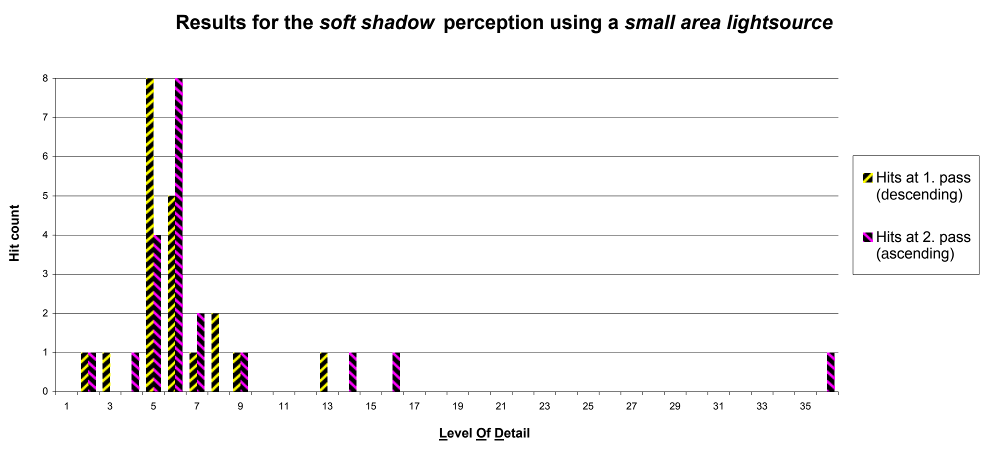


Fig. 4.5: Results of the first two parts of the experiment for an area light source casting soft shadows.

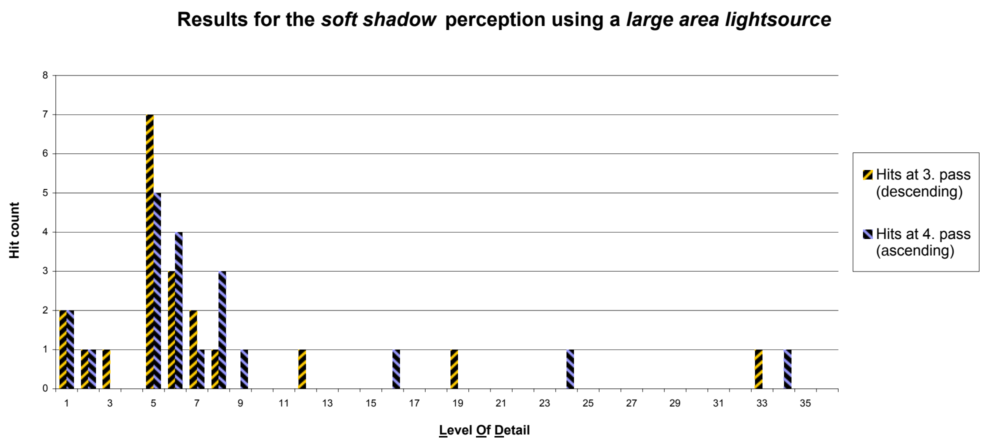


Fig. 4.6: Results for an increased size of the area light source.

mesh to 0.019 seconds using LOD=17 with 7000 triangles.

4.2.6 Conclusions

The presented experiment shows promising results to exploit the human shadow perception for acceleration of the rendering of shadows in a computer generated image. The first results suggest, that it is possible to use a highly simplified instead of the original model to generate soft shadows. This allows for a high speed-up in rendering times.

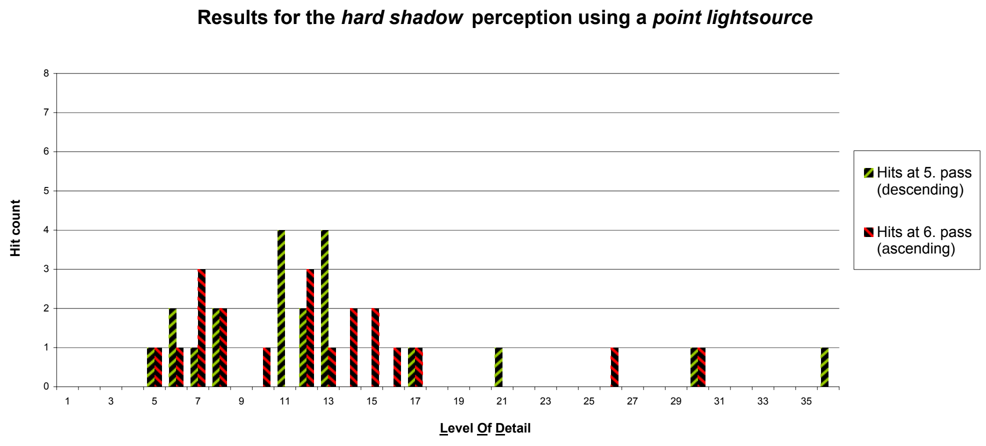


Fig. 4.7: Results for hard shadows cast by a point light source.

As a direction of future work, the generalization to other objects and real-world scenes with a much higher complexity and larger shadow regions could be evaluated. Another topic could be the evaluation of 3D shapes, which one has parametric control over. Also, a nonplanar surface as shadow receiver could be used.

4.3 Self-Shadowing: Static Case

4.3.1 Introduction

In this section a method for real-time shading of folded surfaces such as cloth is described. The surfaces, which possibly contain holes and complex folds are lit under realistic illumination conditions.

The surface is assumed to be given as a static triangle mesh. The appearance of a surface point is given by the radiance leaving the point in the direction of the viewer. According to the rendering equation [Kajiya 1986; Jensen 2001], this radiance is obtained by integrating the incoming radiance over all incoming directions at the vertex v. These directions are typically represented by a hemisphere, $H(v)$, centered around v's surface normal.

4.3.2 Related Work

The problem of shading folded surfaces, especially cloth, was addressed by Stewart [Stewart 1999]. The main idea of his algorithm is a preprocessing step, which computes 3D *visibility cones* for each vertex point, which are used to determine the parts of the environment *seen* by this point. The cones are stored for each vertex and used to evaluate the direct primary irradiance at runtime by doing several intersection tests. The local illumination model described in [Stewart & Langer 1997] is used to calculate the resulting irradiance value.

Stewart reduces the calculation of the 3D visibility cones to a number of 2D visibility computations by slicing a polygonal mesh with parallel planes. If the illumination comes from a point light source, it is sufficient to test whether the point lies in the visibility cone. If a uniform diffuse area light source illuminates the surface, the area light source is intersected with the visibility cone and a contour integral around the boundary of the part of the source inside the cone yields the direct primary irradiance [Shirley 2000]. If the surface is illuminated by a uniform diffuse spherical source that surrounds the surface, a contour integral can be applied to the boundary of the visibility cone in the same manner as that of the area source. Although the results presented by Stewart are convincing, the necessary computation of intersection areas and the evaluation of the integrals prevent the use of complex shaped light sources and changing lighting conditions in real-time.

To overcome these problems the presented method uses *binary visibility maps* instead of the Stewart's visibility cones. These binary visibility maps are computed in a preprocessing step as follows: a finite set of directions on the hemisphere

$H(v)$ belonging to a vertex v is considered. For each such direction it is determined whether the environment is visible or occluded by the surface, and the binary value at the corresponding position in the visibility map is set accordingly. Following the work of Stewart [Stewart & Langer 1997] the incoming light from directions occluded by the surface itself is neglected. Only light from directions in which the outside environment is visible contributes to the radiance of a surface point.

To discretize the directions three different models are evaluated: a hemicube, a single plane and a subdivision of the hemisphere into rectangles using spherical coordinates. This way, the algorithm can accurately handle all extended light sources whose projection onto the hemisphere, the hemicube or the single plane can be approximated with sufficient accuracy by the visibility map. To illuminate the surface realistic lighting conditions are encoded in a global environment map. During the rendering process, the radiance values stored in this environment map are used to calculate the outgoing radiance in direction of the viewer for each vertex of the mesh. For the computation various reflectance models of the surface can be applied.

4.3.3 Algorithm Outline

This section gives a brief outline of the algorithm:

- Preprocessing
  - For each vertex v of the mesh the visibility map is computed by rendering the scene using v as eye point and the normal n of the mesh in v as viewing direction. Pixels of the visibility map not covered by the mesh encode their corresponding direction in the hemisphere.
  - The visibility maps are computed and stored for each vertex.
- At runtime
  - The radiance with respect to the observers' position is calculated for each vertex using a standard reflection model. The vertex´ visibility map is used to determine whether the radiance from a certain direction contributes to its radiance value.
  - The positions of point light sources are transformed into the coordinate system of the visibility map. It is subsequently decided whether the point light source contributes to the illumination of the vertex or not. This can easily be performed using the visibility map.

– Finally, the calculated radiance values are assigned to their corresponding vertices and Gouraud interpolation is performed.

4.3.4 Preprocessing

During the preprocessing phase the visibility maps for each vertex of the mesh are generated. Figure 4.8 shows details of the computation.
Three different ways to encode a visibility map are compared: the hemicube [Cohen & Greenberg 1985], a single plane [Cohen & Wallace 1993] and the hemisphere discretized into a rectangular grid. To obtain the visibility map in a vertex v, the mesh is projected by a central projection with center v to one of these models (see Figure 4.9).

The hemicube, the hemisphere and the single plane are centered around v's surface normal n. The single plane is oriented perpendicularly to n. If the environment cannot be seen in a certain direction, the visibility of this direction in the visibility map is set to the RGB value $(0, 0, 0)$, otherwise the direction itself is encoded in an RGB value.

Because a cube model of the environment map is used, the RGB value (x, y, n) is given by the x and y coordinate in the picture of the n-th side of the cube of the environment map. The numbering of the sides is as follows: The top face is numbered 0, the bottom face 5, the front face 1, the right face 2, the back face 3 and the left face 4.

Figure 4.8 shows a sample mesh and the corresponding visibility map for one vertex. A hemicube model is used with a resolution of 64×64 for the top and 64×32 for the sides. The top image shows the mesh with a vertex (blue dot) inside a fold. The vertex is marked by its red normal.

The bottom image shows the unfolded hemicube for the above mentioned vertex. For simplicity in this picture the directions in which the environment can be seen are coded by black and not by the direction itself. The directions where the environment cannot be seen are drawn in green. The red dot shows a projected point light source. In this example the light source can be seen by the specified vertex. Therefore the vertex is lit.

The resolution of the visibility maps is $n \times m$. Other resolutions tested lie in $n, m \in [4, 256]$. The presented method allows the resolution to be defined by the user. Good results are achieved using $n \times m \geq 64$, as the possible resolution of the outgoing radiance in a vertex is limited by this number in case of diffuse lighting

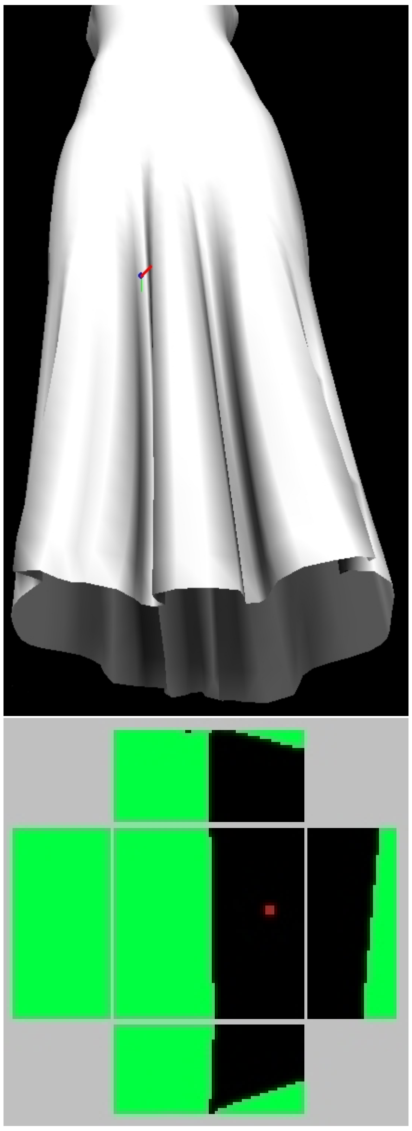


Fig. 4.8: Visibility test. The top image shows the mesh with a vertex (blue dot) inside a fold. The vertex is marked by its red normal.
The bottom image shows the unfolded visibility map of this vertex. The model used for the visibility map is a hemicube with a resolution of 64×64 pixels for the top side of the cube and 64×32 for the sides.
For simplicity, in this picture the directions in which the environment can be seen are coded by black and not by the direction itself. The directions where the environment cannot be seen are drawn in green. The red dot shows a projected point light source. In this example the light source can be seen by the specified vertex. Therefore the vertex is lit.

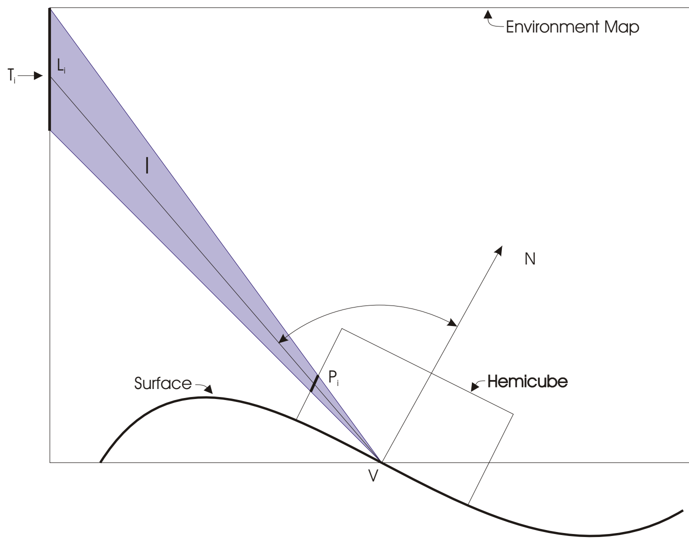


Fig. 4.9: The mapping between the *binary visibility* map of vertex v and the global environment map T_i is encoded in each pixel P_i of the visibility map itself. The environment patch T_i emits L_i in the direction I. θ is the angle between the surface normal N in v and the direction I of the incoming radiance. For simplicity, only a side view is shown.

of the environment. For a resolution less than 64 this results in blotchy images.

The central projection for all three models can be computed using standard ray tracing. For the hemicube and the single plane standard OpenGL rendering can be applied. However, using standard OpenGL rendering to project the mesh onto the sphere is complicated and involves specific hard to implement clipping steps.

For a static mesh the visibility map is fixed and is stored together with the coordinates and the material parameters (reflectance parameters) of a point.

4.3.5 Comparison of Rendering Methods

Let **v** be the vertex of the mesh, the visibility map is determined for. The following rendering methods for generating the visibility maps are evaluated:

1. *OpenGL-Rendering*: The mesh is rendered using a triangle stripped display list of all triangles.
2. *OpenGL-Rendering with triangle pre-selection*: In a first step, all triangles of the mesh are sorted into a three-dimensional grid. Then, triangle strips are

Method	Model	Resolution	Running time (sec)	Distance (ums)
Triangle Rasterizer	HC	8x8	28.4	2.0
	HC	16x16	30.0	
	SP	8x8	4.3	
	SP	16x16	5.0	
	HS	16x4	51.5	
	HS	32x8	54.7	
Triangle Rasterizer with triangle preselection	HC	8x8	15.2	1/6
	HC	16x16	17.0	
	SP	8x8	3.1	
	SP	16x16	3.7	
	HS	16x4	15.0	
	HS	32x8	20.6	
Raytracer with triangle preselection	HC	8x8	120.0	1/6
	HC	16x16	457.9	
	SP	8x8	32.2	
	SP	16x16	104.3	
	HS	16x4	33.3	
	HS	32x8	110.6	
Raytracer with grid traversal	HC	8x8	27.1	2.0
	HC	16x16	103.8	
	SP	8x8	9.3	
	SP	16x16	46.1	
	HS	16x4	7.2	
	HS	32x8	26.8	

Tab. 4.1: Overview of the running times of the visibility calculation routines. Abbreviations used: HC= hemicube, SP= single plane, HS= hemisphere. UMS=units of mesh size.

generated for all triangles contained in a certain cell using a straightforward stripping algorithm. Finally, the triangle strips are stored in display lists. During the rendering of the mesh, the display lists of only those voxels are called, that are within a given distance from the vertex **v**. This distance is given in units of the maximum dimension of the mesh.

3. *Raytracer*: For every discretized direction of the visibility map all triangles of the mesh are tested for an intersection with the ray leaving the point in that direction.

4. *Raytracer with triangle preselection*: Similarly to OpenGL rendering with triangle pre-selection, all triangles of the mesh are sorted into a three-dimensional grid and only triangles in grid cells close to **v** are tested.

5. *Raytracer with grid traversal*: only triangles lying in grid cells passed by the ray are tested.

Table 4.1 summarizes the run times for all combinations of methods and models.

The computations were performed on an Intel Celeron 800 MHz machine with a NVIDIA TNT2 graphics card. To evaluate the core speed of the visibility calculation routines, the rendering of the environment map is not included into the measurements, that is, no directions are encoded.

The table demonstrates quite clearly that OpenGL rendering with triangle preselection in singleplane mode and raytracer with grid traversal in hemisphere mode are the fastest techniques for the preprocessing step.

The singleplane model is efficient, however it has the disadvantage that the aperture angle must be less than 180 degrees, so only part of the half space is evaluated. Therefore, the model may miss some small folds.

Using the hemicube model is relatively slow compared to the singleplane model, because five pictures must be rendered for each point (the other models only need to render one image). On the other hand, it is more accurate than the single plane model and in contrast to the hemisphere it can be hardware-accelerated.

The table also shows that the OpenGL runtime only slightly depends on the resolution of the images that are generated. The situation is radically different in case of the raytracer, the double resolution needs twice as many rays.

4.3.6 Realtime Rendering

During the real-time rendering the outgoing radiances have to be computed for every vertex of the mesh.

Illuminating the surface using an environment map

The environment is stored in a cube map. For the result images the 24bit RGB pictures are generated by hand. For real applications they can be generated using high dynamic range images from real world environments. The resolution of the environment map is adapted to the resolution of the visibility map in such a way that one texel in the environment map corresponds to approximately one pixel of the visibility map.

Calculating the outgoing radiance

The outgoing radiance in vertex v at the surface location x in direction of the viewer has to be computed.

According to the rendering equation [Kajiya 1986] the amount of incident light reflected towards the viewer has to be gathered. For this purpose, the incident radiance of each pixel is weighted by the Δ-form factor of the pixel itself and then used as incoming radiance of a reflection model.

The Δ-form factors are derived from the rendering equation as following [Jensen 2001]:

$$L_o(x,\vec{\omega}) = L_e(x,\vec{\omega}) + \int_S f_r(x, x' \to x, \vec{\omega}) L_i(x, x' \to x) V(x,x') G(x,x') dA' \tag{4.1}$$

with:

L_o: outgoing radiance $[Wm^{-2}sr^{-1}]$
L_e: emitted radiance $[Wm^{-2}sr^{-1}]$
f_r: BRDF $[sr^{-1}]$
L_i: incident radiance $[Wm^{-2}sr^{-1}]$
$\vec{\omega}'$: incidence direction
$\vec{n}$: normal at the surface location x
x': another surface location
$\vec{n}'$: normal at x'
dA': differential area at x'

$(x' \to x)$: radiance leaving x' in the direction towards x
S: set of all surface points
The visibility between two points is defined by:

$$V(x,x') = \begin{cases} 1 & : \quad x \text{ and } x' \text{ are mutually visible} \\ 0 & : \quad \text{otherwise} \end{cases} \tag{4.2}$$

and a geometry factor is introduced:

$$G(x,x') = \frac{(\vec{\omega}' \cdot \vec{n}')(\vec{\omega}' \cdot \vec{n})}{\|x' - x\|^2} \tag{4.3}$$

For the evaluation of the hemicubes it is assumed, that every surface in the model is Lambertian, so that the reflected radiance is constant in all directions. This reduces equation (4.1) to:

$$\begin{aligned} B(x) &= B_e(x) + \int_S f_{r,d}(x) B(x') V(x,x') G(x,x')\, dA' \\ &= B_e(x) + \frac{\rho_d(x)}{\pi} \int_S B(x') V(x,x') G(x,x')\, dA' \end{aligned} \tag{4.4}$$

B: radiosity (outgoing) $[Wm^{-2}]$
ρ_d: diffuse reflectance for a Lambertian surface ($\rho_d = \pi f_{r,d}(x)$)

Discretizing:

$$B_i = B_{e,i} + \rho_i \sum_{j=1}^{N} B_j F_{ij} \tag{4.5}$$

The form factor F_{ij} from differential area dA_i to differential area dA_j is

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{V(x,x') G(x,x')}{\pi}\, dA_j\, dA_i \tag{4.6}$$

Delta form factors for the hemicube pixels: (A hemicube pixel covers the area ΔA, the visibility information is encoded into the pixels):

$$\Delta FF = \frac{G(x,x')}{\pi} \Delta A \tag{4.7}$$

For the hemicube model the Δ-form factors for top and side faces are computed analog to [Cohen & Greenberg 1985].

The contribution of one pixel in the visibility map of x is computed using the radiance stored in the corresponding texel of the environment map. This radiance is weighted by the Δ-form factor of the pixel and then used as input of a local illumination model, which is Lambertian reflection.

For real-world environment maps no Δ-form factor has to be applied to the outgoing radiance stored in the environment map, since it is already encoded in the corresponding real-world picture.

Due to the superposition property of light the total amount of radiance leaving the vertex in direction of the viewer is then easily obtained by summing the contributions of outgoing radiance of all pixels of the visibility map not marked as occluded.

The incident radiance corresponding to a pixel is taken from the environment map using the texel of the environment map encoded in the visibility map, see Figure 4.9.

The Δ-form factors have to be computed only once and can be reused for every vertex. After calculating the radiance values for all vertices, the mesh can be rendered using a standard OpenGL-Renderer with Gouraud interpolation.

For the point light source any desired illumination model can be incorporated into the algorithm.

Dynamic environment maps

Due to the above calculations, the algorithm allows the dynamic modification of the illumination condition in realtime by using different cube maps. For example, if the user wants to rotate the surface in the environment, a rotated cube map is generated.

4.3.7 Results

Figure 4.10 and 4.12 show the static mesh of a cloth illuminated by virtual light encoded in hand made cubic environment maps (Figures 4.11 & 4.13). Additional videos are also available.

Video V

The environment maps faces represent uniformly emitting light sources. In all pictures, the relative position of the model with respect to the environment map is fixed. In order to visualize the effect of the different lighting conditions, the whole

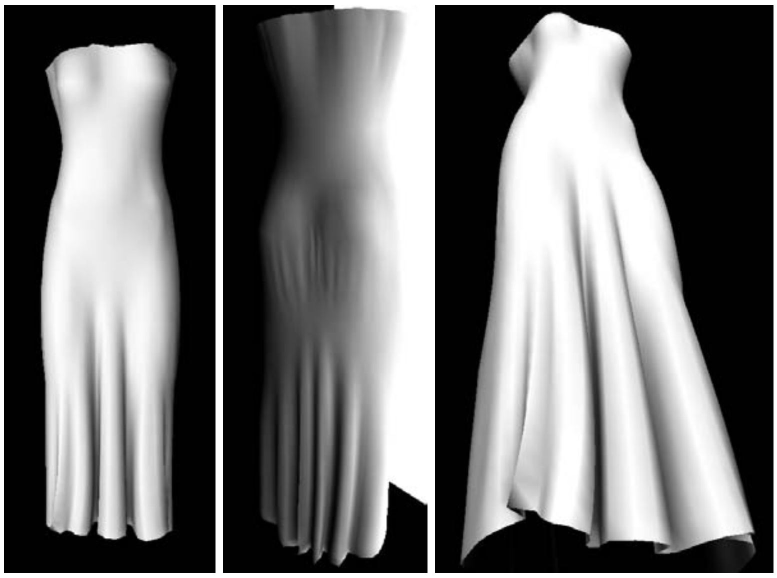


Fig. 4.10: These three images show a folded dress consisting of 3120 vertices. The left image shows a frontal view of the dress. Due to the pre-computed shadowing the folds in the lower part of the dress are clearly visible. In the center image, the back of the dress is shown, slightly rotated against the front side of the environment cube. The right image is rendered using another point of view, showing the folds in more detail.

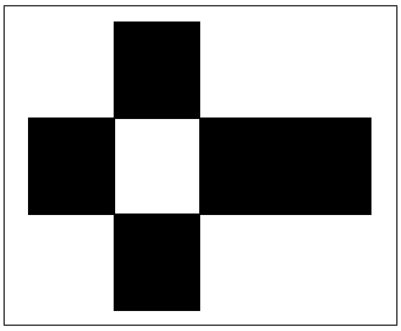


Fig. 4.11: Environment cube map used for the rendering of the above images in Figure 4.10. Only the front side is white.

scene including the environment map is rotated.

The radiance especially in the foldings descents from the illuminated to the dark side. Folds pointing towards the light source are fully illuminated. The situation is even more apparent if the faces of the environment map are treated as colored light sources, see Figure 4.13.

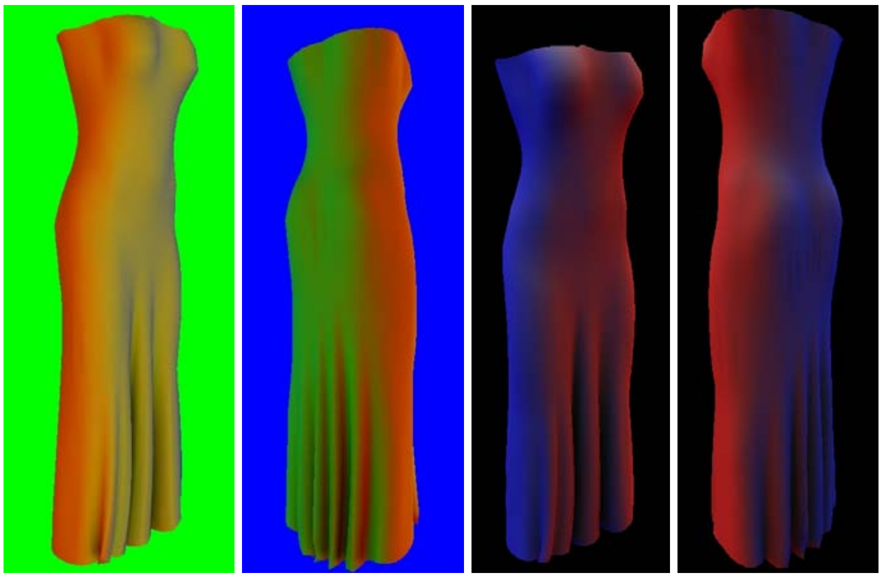


Fig. 4.12: These images show the effects of the usage of the environment faces as colored light sources. The mesh used, is the same as in Figure 4.10. The corresponding environment maps are shown in Figure 4.13. The reflection of the different light sources can be distinguished from each other in the folds.

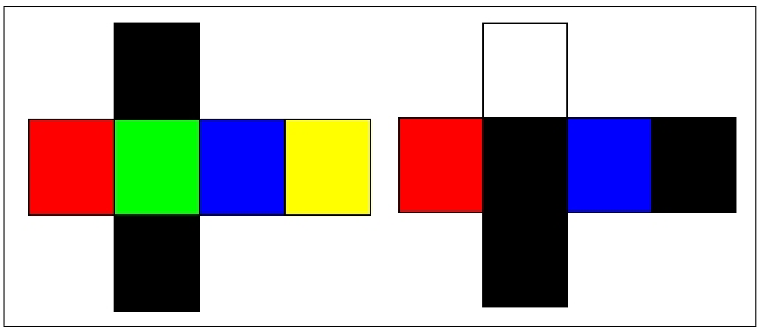


Fig. 4.13: Environment cube maps used for the rendering of the above images. For the two left images top and bottom are black. Sides are red, green, blue and yellow. For the two right images left and right are blue and red, the top is white and the rest is colored black.

Figure 4.14 shows a textured mesh of parts of the Grand Canyon based on satellite altitude data with 66049 vertices. In the left image pre-computed shadows calculated with a hemicube model are used. In the right image no illumination is used. The visual impression of the left images reveals a greater depth impression due to the self-shadowing in the faults.

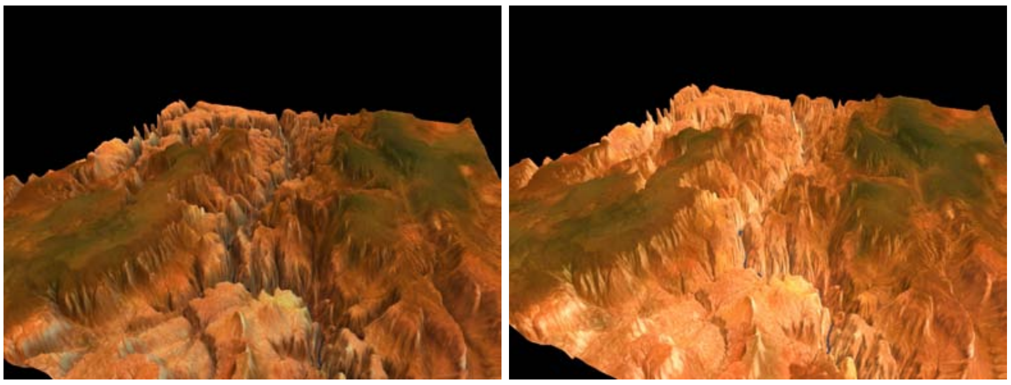


Fig. 4.14: These images show a textured mesh of parts of the Grand Canyon based on satellite altitude data with 66049 vertices. In the left image pre-computed shadows calculated with a hemicube model are used. In the right image no illumination is used. The visual impression of the left images reveals a greater depth impression due to the self-shadowing in the faults.

Due to the vertex based shading it is necessary that the resolution of the triangle mesh provides sufficiently high fidelity. An additional point light source is used in Figure 4.15 with the Utah teapot mesh. The spout casts a shadow on the pot (left image).

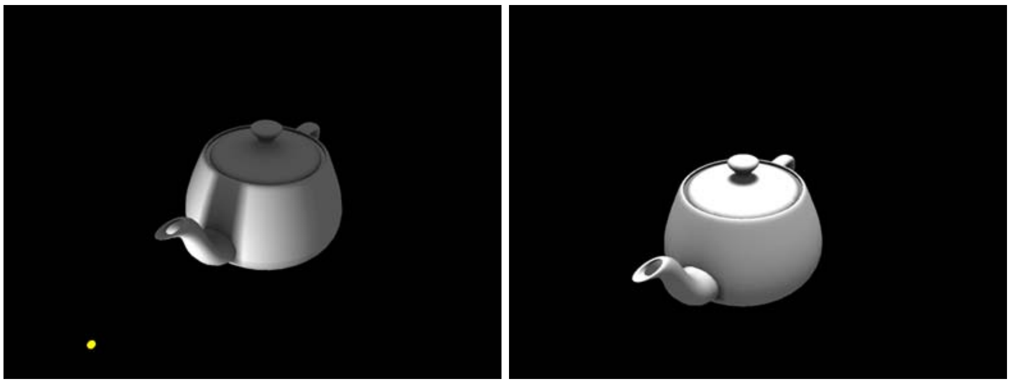


Fig. 4.15: The Utah teapot. The mesh consist out of 3907 vertices and is illuminated with a point light source (yellow dot) in front of the teapot (left image). The light source visibility is calculated on a per vertex basis, as described in the paper. The right image shows the teapot with the light source moved above it. Self shadowing is also visible in both images.

The images in Figure 4.16 show a spaceship consisting of 18720 vertices. Self-shadowing is visible in the propulsion units, at the cockpit and under the wings.

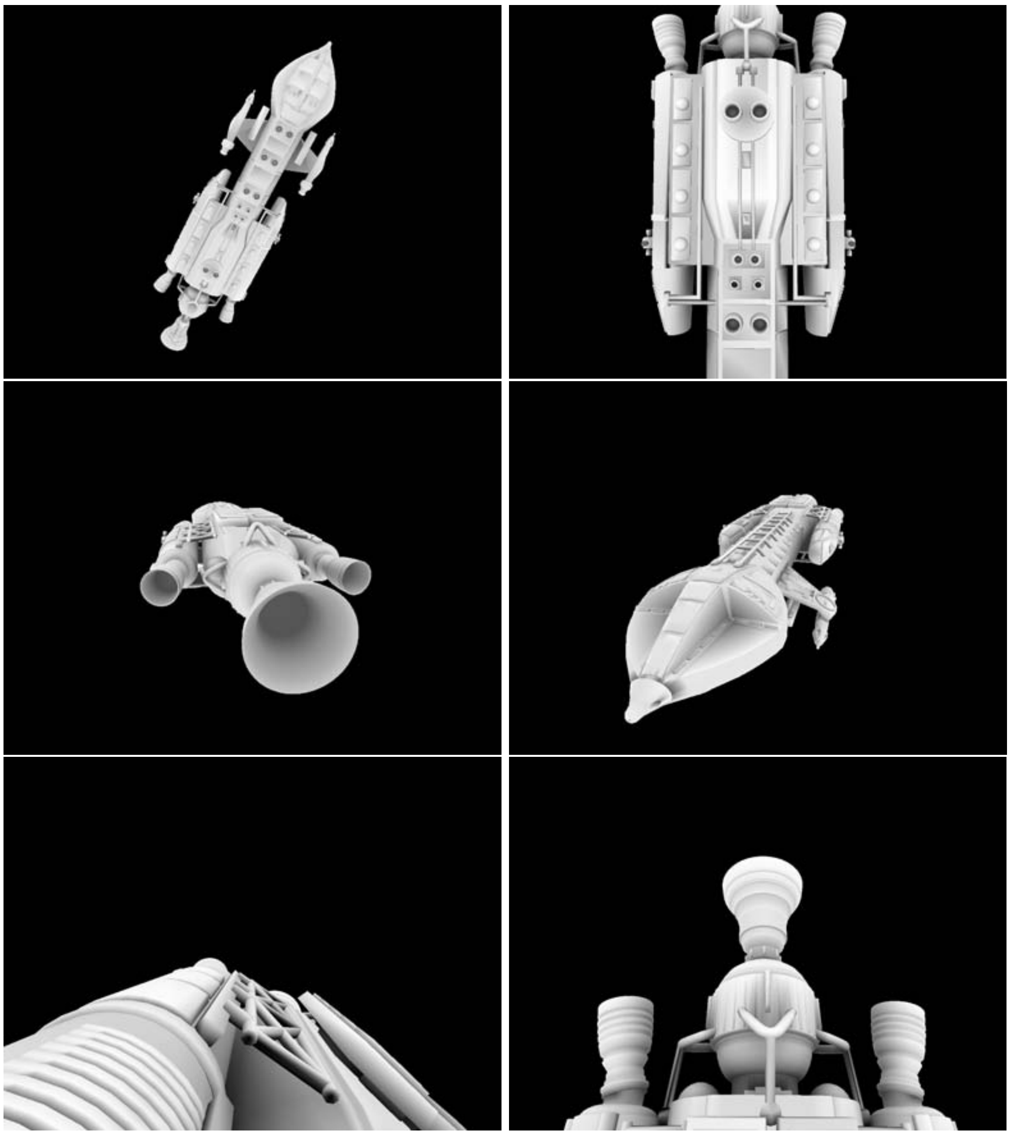


Fig. 4.16: These images show a spaceship model with 18720 vertices, provided by www.3DCAFE.com. A hemicube model with a resolution of 64×64 pixels for the top of the cube and no additional point light sources were used. Self-shadowing is visible in the propulsion units, at the cockpit and under the wings.

4.3.8 Conclusions

The algorithm described in this section is able of illuminating folded surfaces with extended light sources in realtime. The illumination conditions can be changed at runtime.

So far, the meshes cannot be deformed in realtime. This would require a complete new set of visibility maps at every frame. At the moment, the preprocessing step needs at least about 3 seconds of runtime, using the hemiplane model, which is not enough to achieve interactive frame rates.

A possible optimization could be to update only the parts of the mesh which have changed. Furthermore, it might be possible to further exploit graphics hardware acceleration for the preprocessing step, using a hemi-cube model. This enhancement is described in combination with the BTF rendering in Chapter 5.

Moreover, a local illumination model described by Stewart and Langer [Stewart & Langer 1997] can be applied to estimate secondary irradiance. The use of this model yields perceptually acceptable shading without resorting to an expensive global illumination step.

4.4 Self-Shadowing: Dynamic Case

4.4.1 Introduction

Shadows are one of the most important visual clues about the spatial structure of an object. For example for virtual reality applications, such as life-sized cloth visualization, or medical surgery planning, shadows are important because they increase the presence of the virtual objects and the overall realism.

However, the shadows need to be computed at interactive frame rates, otherwise usability and presence will break down. While local illumination models are the strength of modern graphics hardware, more advanced techniques, which include for example soft shadows and ambient occlusion, are notoriously hard to perform efficiently. On the other hand, recent methods like pre-computed radiance transfer [Sloan *et al.* 2002] enable complex illumination effects at fast frame-rates, but are limited to static objects or pre-defined animations due to their long precomputation times.

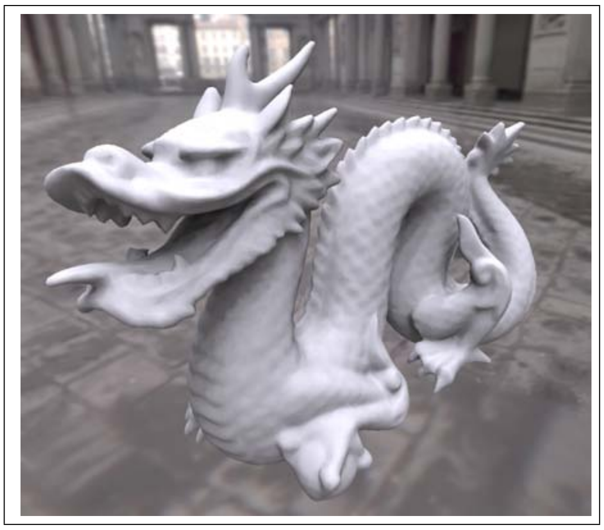


Fig. 4.17: *Stanford dragon* under high-dynamic range illumination including ambient occlusion, which has been computed on graphics hardware.

This section introduces a novel method to efficiently compute the visibility for many light directions for each vertex on the GPU at interactive frame rates, as described in [Sattler *et al.* 2004a]. The method is capable of handling large objects and also includes self-occlusion as well as occlusion caused by other objects. This is done by using hardware occlusion query results from vertex fragments as seen from a number of light directions. All geometry in the scene can deform and move, and the illumination can change at no extra cost. Thus, the method computes and renders a first-order approximation of the rendering equation [Kajiya 1986] for opaque, polygonal objects on the graphics hardware.

The algorithm features the following advantages:

- no precomputation, no complex data structures or special preprocessing is needed
- handles arbitrary deforming geometry
- fast hardware-accelerated occlusion calculation.
- high-dynamic range image based illumination
- easy implementation.

4.4.2 Related Work

A vast amount of work has been done on shadow algorithms. Shadow maps [Williams 1978] or shadow volumes [Crow 1977] and all derivates are the classical approaches for point-like light sources. A good overview of algorithms which produce soft shadows can be found in Hasenfratz et al. [2003].

All these algorithms cannot efficiently be used for arbitrary lighting environments. On the other hand, environment mapping as introduced by Blinn et al. [1976] is able to render reflections of incident lighting, but without shadows. Ray tracing is capable of handling globally illuminated scenes, but is naturally limited to the current camera position, Interactive rates are only achieved in a massive parallel environment with optimal acceleration structures [Wald *et al.* 2003b; Wald *et al.* 2003a], which take several seconds to build. Other theoretical work [Purcell *et al.* 2002] on GPU-based raytracing is not yet available in hardware. Very recently there have also been approaches to solve radiosity on graphics hardware [Coombe *et al.* 2004], with interactive rates for small scenes.

To evaluate the illumination received by a point on the surface, there exist mainly two approaches. The first category gathers the incoming radiance at each surface point or vertex and scales with that count. The hemisphere defined by the vertex normal is sampled in different ways. Either rays are shot into predefined directions using standard ray tracing methods, or hemicube [Cohen & Greenberg 1985] sides are rendered using the standard pipeline [Sattler *et al.* 2003]. With ray casting, intensive intersection tests have to be calculated and acceleration structures have to be maintained (for example space partitioning), whereas hemicubes cannot yet be evaluated efficiently on the graphics hardware. Other methods compute visibility cones [Stewart 1999], blocker maps [Hart *et al.* 1999], obscurance maps [Zhukov *et al.* 1998; Iones *et al.* 2003], visibility maps [Neulander 2003] or radiance transfer [Sloan *et al.* 2002] which also includes inter-reflections.

All these methods require certain amounts of pre-computation time, and are therefore not suitable for dynamic objects. Very recent work was presented by Kautz et al. [2004]. It is mainly based on fast hemicube rasterization in order to detect blocker triangles. Downsampling of the visibility mask and a coarser blocker mesh are used for speed up. Interactive frame rates are achieved for small animated models. In contrast to their work, the following described method does not need a mesh hierarchy nor any additional graphics memory during run-time.

The second category is based on the approximation of the ambient environment by point or directional light sources, which amounts to reversing the first approach from *inside-out* to *outside-in*. That is, the visibility computation is originated at the light sources. Lately, NVIDIA proposed a hardware-accelerated 2-pass method, using accumulated shadow maps [Pharr 2004; Randima 2004], which is also used in many shaders in commercial rendering software packages [Landis 2002].

To minimize sampling artifacts, jittering of the depth maps is introduced. This approach involves common shadow mapping projection problems [Kilgard 2002]. To achieve usable results, several seconds per frame are needed. Changing to a new viewpoint requires new shadow mapping passes or an unwrapping process to obtain an occludence texture. Superimposing images was also done by Keller et al. [1997] to compute instant radiosity. This also requires space-partitioning structures.

4.4.3 Ambient Occlusion Calculation

In this section, the core of the new method is presented. In the following, a triangle mesh with vertex normals is given. To compute the outgoing radiance L_r at a surface point $\mathbf{x}$ into direction $\mathbf{v}$ the following equation has to be evaluated:

$$L_r(\mathbf{x},\mathbf{v}) = \int_{\Omega} f_r(\mathbf{x},\mathbf{v},\mathbf{l}) L_i(\mathbf{x},\mathbf{l}) V(\mathbf{x},\mathbf{l})(\mathbf{n}_x \cdot \mathbf{l}) dl$$

where Ω is the hemisphere domain over $\mathbf{x}$, f_r the BRDF, L_i the incident radiance from direction $\mathbf{l}$, $V(\mathbf{x},\mathbf{l})$ the visibility from $\mathbf{x}$ to direction $\mathbf{l}$ and $\mathbf{n}_x$ the vertex normal at $\mathbf{x}$. The integral is discretized by k light directions $\mathbf{l}_j$, $j \in [1, \ldots, k]$, which leads to

$$L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{j=1}^{k} f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}_j) L_i(\mathbf{x}, \mathbf{l}_j) V(\mathbf{x}, \mathbf{l}_j)(\mathbf{n}_x \cdot \mathbf{l}_j)$$

The following sections concentrate on the efficient computation of the term $L_i(\mathbf{x}, \mathbf{l}_j) V(\mathbf{x}, \mathbf{l}_j)(\mathbf{n}_x \cdot \mathbf{l}_j)$.

Overview

The method relies on depth test results from renderings of the desired object. The problem of light direction visibility for each vertex is approached by considering a set of k directional light sources $\mathbf{l}_j$ and determine the visibility of all N vertices at once as seen from each of the light source directions.

Because the number of light source directions k is much smaller than the number of vertices N for large models, the *outside-in* approach is much more efficient than the *inside-out* approach, that is, a much smaller number of render passes, k instead of N, of the object into the depth buffer is needed.

More precisely, all geometry as seen from a light source direction is rendered into the depth buffer. Then, all vertices are rendered again as a point set. An individual occlusion query per vertex allows the algorithm to retrieve those vertices that passed the depth test. For these, the currently considered light source direction is marked as visible and stored in a matrix M, which latter will be called *visibility matrix*. This process is repeated for each light source, updating the appropriate entries in M.

Visibility Matrix Computation

The method makes heavy use of the OpenGL *OcclusionQuery* extension[1] [OSS 2005] . The purpose of this extension is to deliver the number of fragments that passed both depth and stencil test. In contrast to its predecessor[2] it is asynchronous, that is, it does not use a *stop-and-wait* execution model for using multiple queries.

[1]ARB_OCCLUSION _QUERY
[2]HP_OCCLUSION_QUERY

This allows applications to issue many occlusion queries before asking for the result of any one. As mentioned above, in the first pass, the unlit scene is rendered into the depth buffer from one of the light source directions. This is done in orthographic projection mode.

In the second pass, all vertices are rendered as *glPoints* with size 1, without updating the depth buffer. An offset (*glPolygonOffset*) with default values $(1.0, 1.0)$ is used to avoid rounding issues. Each single vertex $i \in N$ is handled by an individual occlusion query and the visibility matrix M, which stores visibility information for each vertex i to the light direction $\mathbf{l}_j$, is updated for all j.

For later performance reasons, not a single boolean visibility bit is stored, but instead the dot product computed from the vertex normal $\mathbf{n}_i$ and the vector defined by the light source direction $\mathbf{l}_j$ if the vertex is visible from that direction. $\mathbf{l}_j$ is computed once for each virtual light source direction. Therefore the following matrix entries are obtained:

$$M_{ij} = \begin{cases} \mathbf{n}_i \cdot \mathbf{l}_j & : \text{vertex visible} \\ 0 & : \text{vertex invisible} \end{cases}$$

Figure 4.18 gives an overview of the core algorithm as pseudo-code.

Rendering

After M_{ij} has been computed, the final color c_i for each vertex i can be computed as:

$$c_i = \sum_{j=1}^{k} M_{ij} I_j$$

where I_j is the 3-component (RGB) color of the light coming from direction $\mathbf{l}_j$. Figure 4.19 shows a simplified data flow of the approach. Note, that neither M_{ij} nor I_j change, if the viewpoint is changed.

Creating the Lightsphere

To approximate ambient occlusion with single directional light sources, k light directions have to be distributed. Each light direction is represented by a point on the unit sphere. The distribution of points on the sphere should satisfy the following conditions. The visibility computation should be done only once and the illumination environment should be changeable without doing new queries (see subsection *Image Based Illumination*). Furthermore, to allow an easy increase of the number of light directions, already computed parts of M should be re-usable.

```
enable orthographic projection
disable framebuffer
for all light directions $j$ do
    set camera at light direction $\mathbf{l}_j$
    render object into depth buffer with polygon offset
    for all vertices $i$ do
        begin query $i$
        render vertex $i$
        end query $i$
    end for
    for all vertices $i$ do
        retrieve result from query $i$
        if result is „visible“ then
            $M_{ij} = \mathbf{n}_i \cdot \mathbf{l}_j$
        end if
    end for
end for
```

Fig. 4.18: Outline of the core algorithm for visibility matrix calculation.

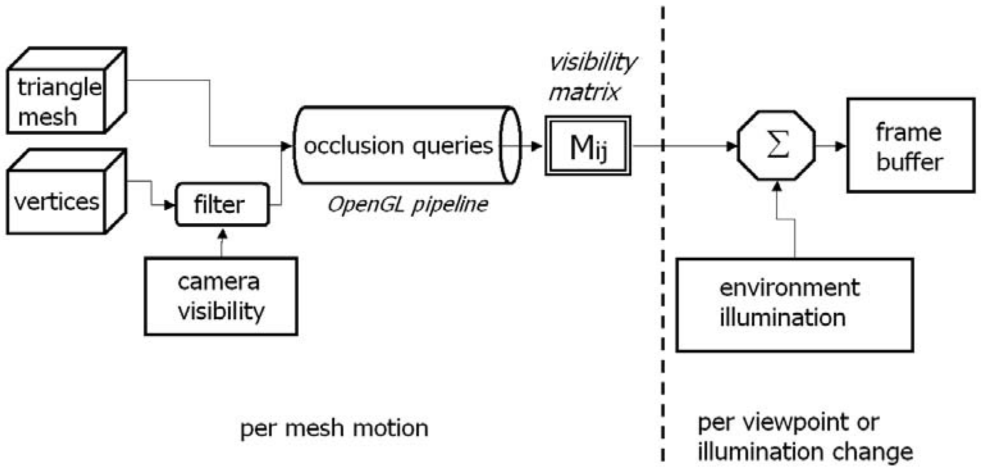


Fig. 4.19: Simplified data flow of the method.

Therefore, the distribution should equally sample the environment in all directions. Recent approaches like [Agarwal *et al.* 2003; Kollig & Keller 2003] which do an efficient sampling of the environment map are not easily adaptable for this approach. They need seconds to minutes of preprocessing time to reduce the number of light directions, therefore would not allow interactive change of the lighting environment. For equal distribution of points on a sphere, several methods exist [Fejes Toth 1972; Whyte 1952; Saff & Kuijlaars 1997].

The proposed algorithm uses the following preprocessing procedure, based on subdivision of a regular solid. As a start the vertices of a unit octahedron are used, for example $k = 6$ light directions at subdivision level $s = 0$. To generate level $s + 1$, midpoint subdivision of the edges on level s is done and the new vertices are projected on the unit sphere, thus creating a polyhedron with $2 \cdot 4^{s+1}$ faces and $2 + 4^{s+1}$ vertices. This structure allows to add new sets of well distributed points, while using the occlusion queries of all coarser subdivision levels. Figure 4.20 shows several increasing configurations.

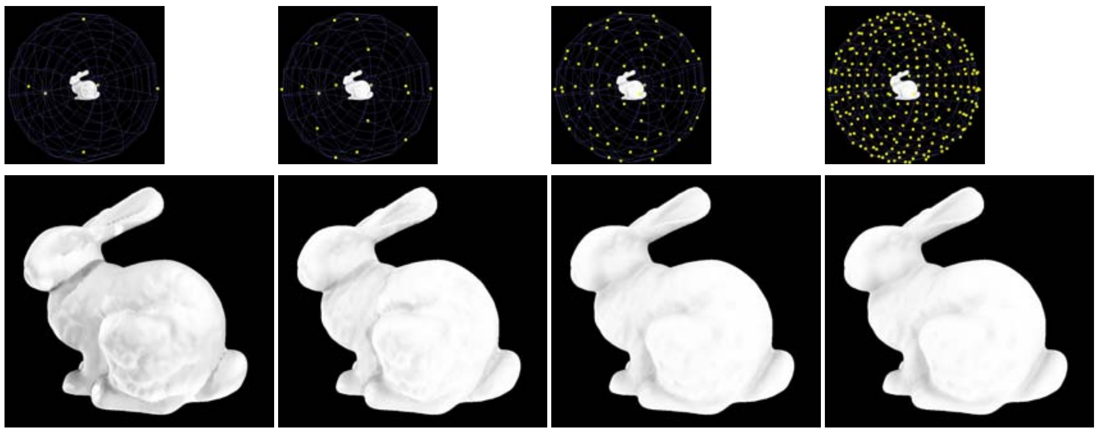


Fig. 4.20: Object rendered with different number of light directions (k=6, 18, 66, 258). The upper row shows the lightsphere configurations, where the yellow dots represent the light directions.

4.4.4 Optimizations

Whenever the object is moved or deformed, that is when vertex positions change, a complete re-computation of M is needed. In the following, several methods are presented, that significantly reduce this effort, so that interactive frame rates can be maintained even under these circumstances. Even for static geometry, depending on the viewpoint, not all vertices must be evaluated.

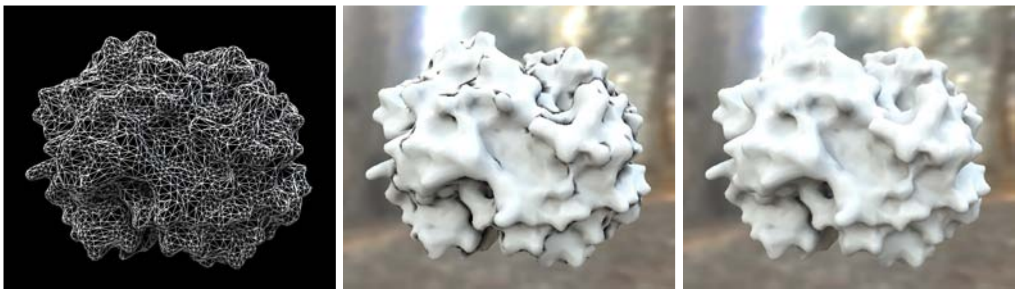


Fig. 4.21: Vertex filtering optimization for the computation of M has to be done carefully, otherwise artifacts will occur (middle). Left: wireframe; right: correct rendering.

Vertex filtering using temporal coherence

The first optimization exploits temporal coherence by observing that during a viewpoint change only a small number of polygons become visible for the first time (at most those that cross the object silhouette). Consequently, the visibility matrix M can be computed lazily, thereby distributing the computation effort over several frames.

More precisely, two lists of triangles are maintained: a list of *unseen* triangles, $\mathcal{U}$, containing all triangles that have not yet been visible and a list $\mathcal{T}$ (*todo*), containing triangles that will be seen in the next frame for the first time. $\mathcal{U}$ is initialized with all vertices once. Vertices which belong to triangles stored in $\mathcal{U}$ have not yet been visible from the camera so far.

Thus, their occlusion information is not needed with respect to the light directions. Vertices which belong to triangles which are stored in $\mathcal{T}$ have to be processed for the next frame, because these triangles will then be visible.

Using the two lists in each frame, the occlusion information has to be determined only for a small fraction of vertices (see also Figure 4.19). In order to compute list $\mathcal{T}$, an algorithm similar to the one in subsection *Ambient Occlusion Calculation* is performed, except that here *triangles* are rendered in the second pass and perspective projection is used.

More precisely, in the first pass, the unlit mesh is rendered into the depth buffer as seen from the new camera position. In the second pass, only the triangles still in list $\mathcal{U}$ are rendered, each with its own occlusion query (again with offset and without depth buffer update). Obviously, when $\mathcal{U}$ contains less triangles than a certain threshold, no performance increase is gained any more. In that case, all triangles from $\mathcal{U}$ are just added to $\mathcal{T}$ and this optimization is skipped in the remaining frames.

Note, that for this optimization triangles, not vertices are considered. Otherwise, artifacts could occur, because a triangle might be visible although one of its vertices is not, and thus its corresponding value in M has not yet been computed. This is illustrated in Figure 4.21 and the performance gains introduced by this optimization are discussed in the results section.

Changing the Lightsphere configuration

A further optimization is to dynamically change the lightsphere configuration. While the scene is animated or objects are deformed, the number of light directions can be decreased. In idle time, in order to converge to the exact solution, that count is increased as shown in the upper row in Figure 4.20.

Note, that if the number k is too small, under-sampling artifacts (left *bunny*) can occur depending on the used environment. In practice, a level of $s = 3$ and often even level $s = 2$ produces reasonable results. The algorithm allows user control through the choice of a desired frame rate or a certain configuration.

4.4.5 Image Based Illumination

In this section, the core algorithm is extended to incorporate image-based illumination [Miller & Hoffman 1984; Greene 1986; Debevec 1998], using high- dynamic range environment maps [Debevec & Malik 1997; Lightprobes 2005]. Similar to other approaches [Heidrich & Seidel 1999; Kautz. *et al.* 2000; McAllister *et al.* 2002] the environment is pre-filtered. Therefore, HDRshop [HDRShop 2005] is used to generate a *latitude-longitude* lookup map out of a cube map and a gaussian blur is applied as a filter as shown in Figure 4.22.

The black dots on the right image are the projected light directions. The environment is now parameterized with (θ, ϕ), with the angle $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$. This is also known as Mercator projection. Thus, a look-up of the intensity from the light coming out of the light direction $\mathbf{l}_j$ can easily be performed. When the environment map is rotated relative to the object, the light directions are multiplied by the same rotation matrix to obtain the new look-up positions. A standard gamma correction is applied to all vertex color values.

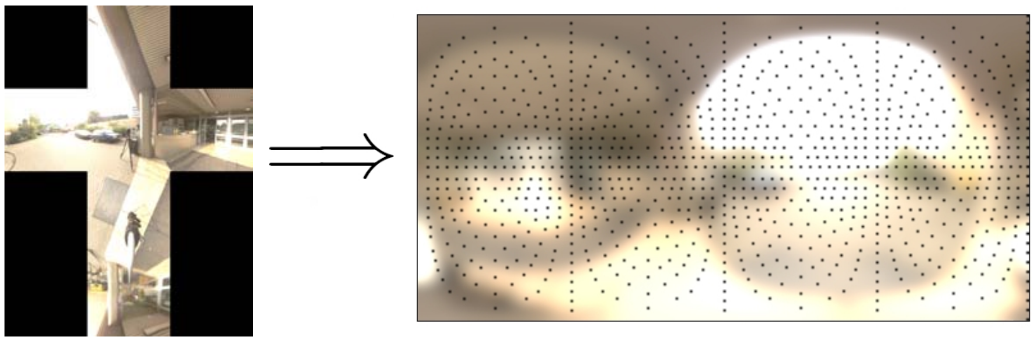


Fig. 4.22: Illumination information stored in a high-dynamic range environment cube-map (left) and filtered version in latitude-longitude representation (right) with projected light source positions (black dots).

4.4.6 Dynamic Geometry and Animations

It should be obvious by now, that in an environment with a single object, a rigid transformation can be handled quite efficiently: a translation can be ignored, while a rotation just amounts to an additional matrix multiplication before the look-up into the visibility matrix. If there are several objects changing positions relative to each other or an object changes shape, than the complete visibility matrix M needs to be re-computed.

However, this can still be done at interactive frame rates, because the algorithm does not need any spatial acceleration structures. Only the list $\mathcal{U}$ (Section 4.4.4) needs to be re-initialized and the vertex arrays for the objects updated. Figure 4.23 shows sample key frames of an animation of a *skeleton* running at interactive frame rates, while changing the viewpoint and the lighting environment. In each frame, M is re-computed according to Figure 4.19.

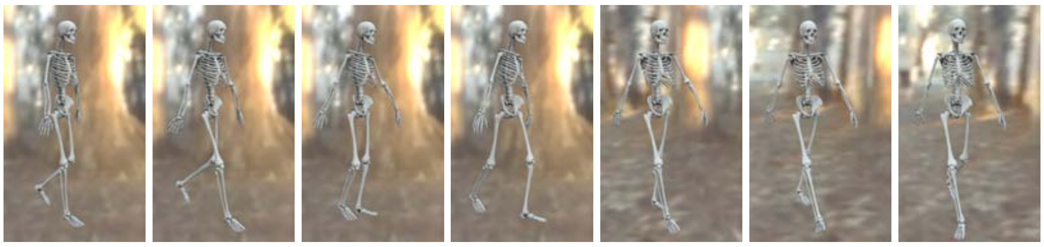


Fig. 4.23: Sample key frames of the *skeleton* animation running at interactive frame rates. A lot of self-shadowing occurs among the bones.

4.4.7 Results

This section presents performance measurements of the algorithm, which was measured on an AMD Athlon64 3200+ (2.0 GHz) under Windows 2000 and an ATI Radeon 9800 XT using OpenGL 1.5. All images and videos were rendered at a resolution of 1280×960 pixels.

Video VI

As for the rendering results, all objects are rendered without textures to make the visual effects of the algorithm more evident. As a matter of course, the method can be combined very easily with texturing or standard shadow mapping techniques [Williams 1978; Kilgard 2002], in order to handle point light sources as well. The left image on Figure 4.24 shows, that the method is also feasible to render large objects for medical visualization. The *teeth* object consists of 116k vertices. Detailed surface structures are visible. The middle and right image in Figure 4.24 show several objects that occlude each other from light directions, which is handled correctly by the method.

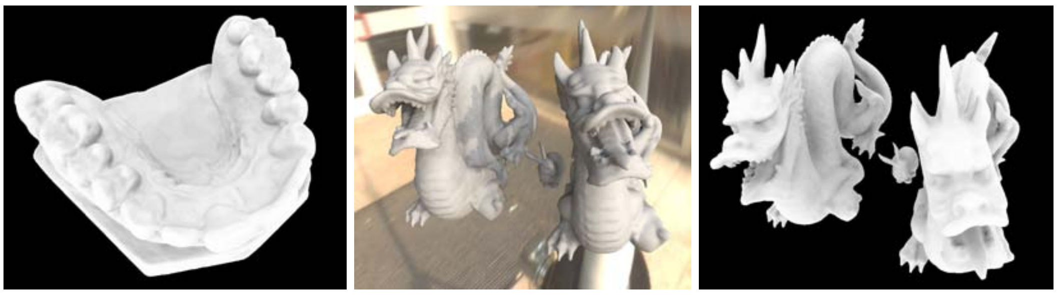


Fig. 4.24: **Left** image: Large object *teeth* (116k vertices) rendered using the new algorithm under homogeneous white illumination. **Middle and right** images: Scene with several objects, rendered under high-dynamic range or homogenous white illumination. Note the darker *bunny* between the *dragons*.

The *bunny* between the *dragons* is much darker, due to the occlusion. Figure 4.25 shows a comparison of the proposed method with OpenGL Phong lighting and a ray-traced image, in a high-dynamic range illumination environment. To achieve the same visual quality 500 samples/ray were needed and it took over half an hour compared to under 1 minute using the new method. The *Igea artifact* consists out of 134k vertices. The slight color shift is due to the slightly different exposures and environment orientations.

Table 4.2 gives an overview over the computation time of the visibility matrix (time$_1$) in milliseconds for several different objects and light directions k. If the

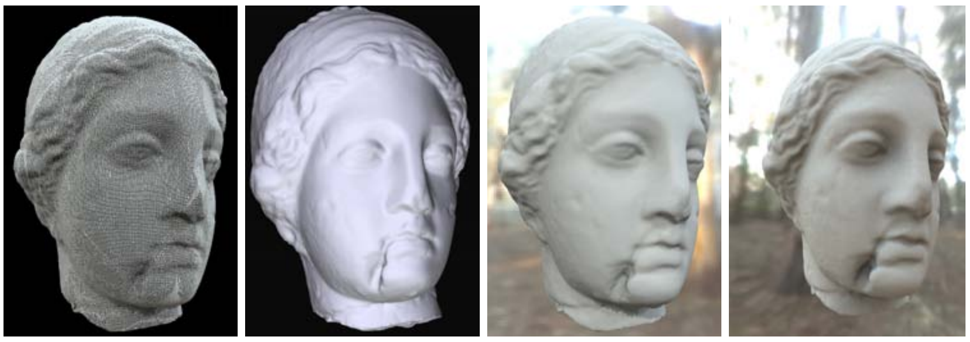


Fig. 4.25: Comparison between different rendering methods of the *Igea artifact* (134k vertices). From left to right: wireframe, OpenGL Lighting, under HDR illumination with the new algorithm and raytraced.

object	vertices	k	time$_1$ (msec)	time$_2$ (msec)	FPS$_1$ (Hz)	FPS$_2$ (Hz)
bunny	35k	6	445	339		
static		18	1082	797		
		66	3606	2666		
		258	13715	10373		
		1026	60304	39765		
teeth	116k	6	1495	1257		
static		18	3651	3041		
		66	12243	10290		
		258	46482	38976		
skeleton	8325	6	133	113	8.08	8.85
animation		18	337	245	3.66	4.10
		66	937	740	1.14	1.35
		258	3301	2472	0.30	0.40
trousers	3219	6	49	34	18.85	22.75
animation		18	103	66	9.23	12.80
		66	330	222	3.03	4.63
		258	1217	811	0.82	1.35

Tab. 4.2: Computation times (*time*) for the visibility matrix for static objects and animations with different lightsphere configurations (k) and frame rates (*FPS*) when rendered. Times with $_1$ are without and $_2$ with vertex filtering enabled. See text for details.

vertex filtering (time$_2$) is enabled, the calculation performance is drastically increased. FPS$_1$ and FPS$_2$ show the frame rates in Hertz with and without the vertex filtering optimization, when the objects are rendered. Real-time frame rates

($>$ 30Hz) for *static* objects and interactive rates for the *skeleton* and pair of trousers *animation* are achieved.

This section presented a new method to calculate vertex-light direction visibility, thus providing a first-order approximation of the rendering equation. This is done by sampling the environment by several directional light sources and efficiently computing vertex visibility from these light directions using hardware-accelerated occlusion queries. In conjunction with vertex filtering optimization, deformable objects and animations can be handled at interactive frame rates.

The proposed method also allows the usage of image based illumination stored in filtered high-dynamic environment maps. The algorithm drastically reduces visibility calculation times and might be incorporated in other visibility determination problems. Because the method is vertex based, artifacts may occur due to undersampling, which is, of course, true for all vertex-based approaches.
As a drawback, the proposed method has to store the visibility list M on the CPU, because the query result is always sent back from the graphics card. It would be a great performance increase, if this result would be available directly on the GPU on future hardware. Additional speed-up could be achieved by allowing parallel query updates through segmented result buffers.

To achieve real-time frame rates for an animation or as a general speed-up, implementation in a parallel environment could be a direction of research.

CHAPTER 5 Material Reflection Properties

5.1 Introduction

Material [MWD 2005]: Etymology: Middle English *materiel*, from Late Latin *materialis*, from Late Latin *materia*
1 : relating to, derived from, or consisting of matter; especially : PHYSICAL *the material world*
2 : of or relating to the subject matter of reasoning; especially : EMPIRICAL *material knowledge*
3 : being of a physical or worldly nature

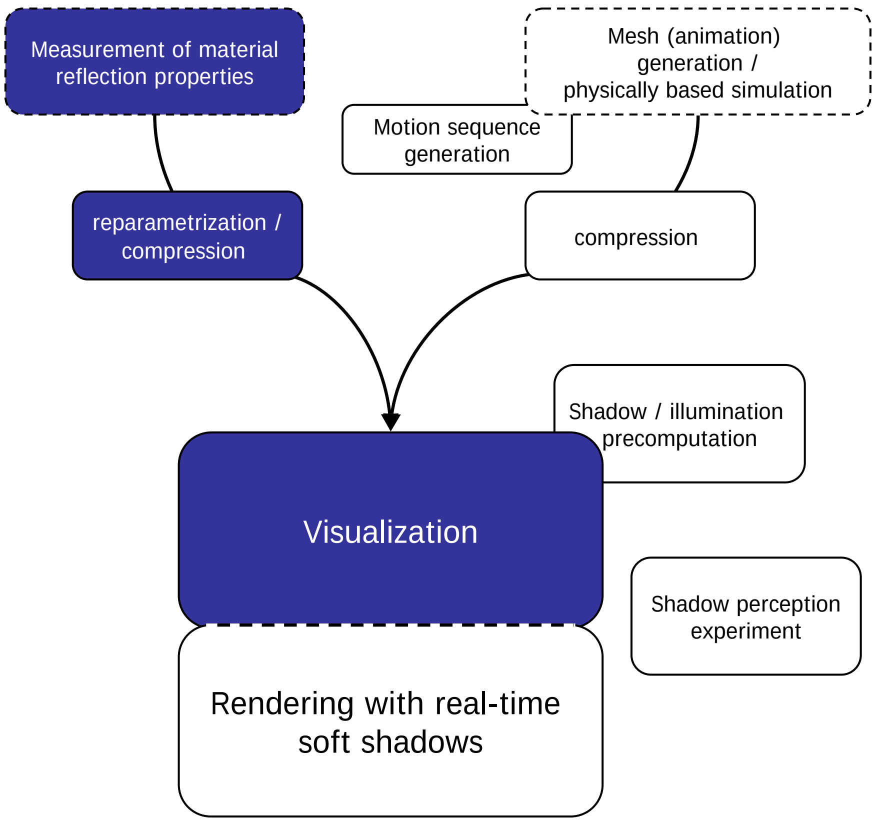

Measurement of material reflection properties
Mesh (animation) generation / physically based simulation
Motion sequence generation
reparametrization / compression
compression
Shadow / illumination precomputation
Visualization
Shadow perception experiment
Rendering with real-time soft shadows

Fig. 5.1: Overview chart with material parts.

The most important part in cloth rendering is the processing of material reflection properties. Within this chapter mayor aspects of this part as highlighted in Figure 5.1 are introduced.

Starting with a detailed description of the acquisition of real-world material surfaces, the complete pre-processing process is described, which allows a fast and semi-automatic processing of different kind of materials.

After that, advanced compression schemes based on principal component analysis are described, to reduce the huge amounts of image data.

Finally, the main topic of this thesis, the visualization aspect of the cloth, is explained. Using programmable graphics hardware in combination with efficient storage and compression schemes, interactive cloth visualization is demonstrated.

5.2 Related Work & Introduction

Efficient and realistic rendering of cloth is of great interest especially in the context of e-commerce. Aside from the simulation of cloth draping, the rendering has to provide the *look and feel* of the fabric itself.

In addition to the microstructure, the mesostructure of a fabric is of great importance for the reflectance behavior of cloth. The mesostructure is responsible for fine-scale shadows, occlusions, specularities and subsurface scattering effects. Altogether these effects are responsible for the above mentioned *look and feel* of cloth.

In this chapter a novel interactive rendering algorithm is presented, which preserves this *look and feel* of different fabrics. This is done by using the bidirectional texture function (BTF) of the fabric, which is acquired from a rectangular probe and after synthesis, mapped onto the simulated geometry. Instead of fitting a special type of bidirectional reflection distribution function (BRDF) model to each texel of the BTF, view-dependent texture-maps are generated using a principal component analysis of the original data. These view-dependent texture maps are then illuminated and rendered using either point-light sources or high dynamic range environment maps by exploiting current graphics hardware. In both cases, self-shadowing caused by geometry is taken into account. An example rendering is shown in Figure 5.2. An animated example can be found on the video.

Video VII

For point light sources, a novel method to generate smooth shadow boundaries on the geometry is also presented. Depending on the geometrical complexity and the sampling density of the environment map, the illumination can be changed interactively. To ensure interactive frame rates for denser samplings or more complex objects a principal component based decomposition of the illumination of the geometry is introduced. The algorithm is also suitable for materials other than cloth, as far as these materials have a similar reflectance behavior.

There are essentially two techniques of cloth rendering according to the way in which mesostructure is captured. The first approach explicitly models the mesostructure of the fabric in detail and renders it using different lighting models and rendering techniques [Gröller *et al.* 1995; Daubert & Seidel 2002] Although these algorithms produce impressive results and some of them are already applicable at interactive frame rates, using these methods, it is difficult to reproduce the special appearance of a given fabric.

In the second approach the reflectance properties of a given real fabric are measured and then used to generate realistic images [Dana *et al.* 1999b; McAllister *et al.* 2002].

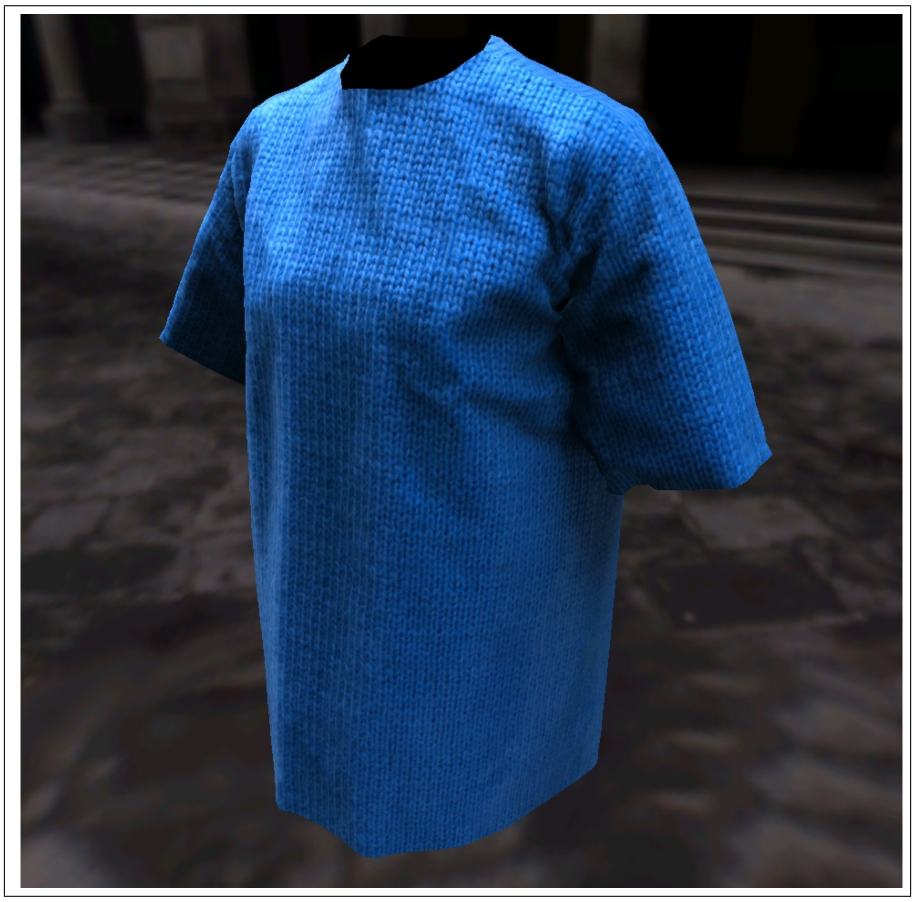


Fig. 5.2: Wool shirt rendered under natural illumination (Uffizi street scene).

As shown by [Dana *et al.* 1999b], the most important optical parameters of opaque materials including their mesostructure can be described by the bidirectional texture function (BTF). This six-dimensional function describes how a planar texture probe changes its appearance when illuminated and viewed from different directions. The resulting texture probes capture all effects caused by the mesostructure like roughness, self-shadowing, occlusion, inter-reflections and subsurface scattering. Furthermore, the BTF describes how the texture has to be filtered when viewed from different directions.

Therefore, in order to achieve the most realistic visualization of a given cloth, the following approach is based on the second approach based on measured BTF data. For the illumination two different methods are provided: first by point or directional light sources. Second, illumination by utilizing high dynamic range environment maps. Both techniques are of interest, since on one hand, illuminating the material by point light sources allows the user to inspect the material under a controlled lighting situation and reveals the mesostructure nicely.

Here, also a new method is introduced, to generate smooth shadow boundaries on polygonal meshes. On the other hand, people can judge and recognize the material more easily under natural illumination than under the simplified and artificial one provided by point light sources. The algorithm uses a decomposition of the illumination of the geometry, to ensure the change of the environment maps at interactive frame rates.

In addition to the mesostructure captured by the BTF, a further essential ingredient for the realistic rendering of cloth are macroscopic shadows caused by self-shadowing of the object. These shadows enhance especially the draping of the fabrics. The main contribution of this algorithm is a realistic real-time visualization of a wide variety of cloth, including highly structured materials like corduroy or knitwear based on measured reflection properties. Special features of this algorithm are

- Preserving the look and feelöf the real cloth.
- Support of point and directional light sources as well as image based lighting at interactive frame rates.
- A simple, but efficient technique to calculate dynamic shadows caused by point or directional light sources with smooth shadow boundaries on polygonal meshes
- A new efficient decomposition technique for illumination of geometry with BTF data, including self-shadowing.

5.2.1 Modeling Mesostructure

Previous work in cloth rendering falls into two main categories. The first is the explicit modeling of the underlying mesostructure and rendering it using volumetric techniques. Modelling has the general advantage of being able to create complete artificial results for non-existing materials. While certain approaches are not real-time capable [Gröller *et al.* 1995; Gröller *et al.* 1996; Xu *et al.* 2001], some interactive methods exist which use special shading models [Daubert *et al.* 2001; Daubert & Seidel 2002]. Up to now, these algorithms are mainly used for knitwear and cannot handle materials like for example corduroy. Image based lighting and macroscopic self-shadowing are neglected.

5.2.2 Measuring reflection properties

Using measured reflection properties of real world surfaces naturally implies higher realism. Effects, which give important visual clues for material identification, like microstructure self-shadowing or scattering are preserved. On the other hand careful measuring is required. Details follow in Section 5.3.

Light fields

Capturing images of models under different lighting conditions and from different viewing angles automatically captures the reflection properties and yields very realistic renderings of the objects, although using these so called light field approaches [Levoy & Hanrahan 1996; Gortler *et al.* 1996; Debevec *et al.* 2000; Chen *et al.* 2002], it is not possible to change the lighting conditions. A general drawback of these approaches is that the measured material properties are coupled with a fixed geometry, thus not allowing to change the geometry or the material without remeasuring the object.

Malzbender et al. [2001] introduced polynomial texture maps, where the coefficients of a biquadratic polynomial are stored per texel, and used to reconstruct the surface color under varying lighting conditions.

Lensch et al. [2001] proposed a method to capture spatially varying materials on known geometry, by finding basis BRDFs for reconstruction on a per-pixel level. These approaches can also be applied for cloth.

BRDFs

BRDFs are four dimensional functions and were introduced by Nicodemus [1970]. These functions describe the reflection distribution at a surface point depending on incoming and outgoing light directions. BRDFs overcome the limitations of geometry coupling, fixed lighting and viewing directions. Early results approximated a single BRDF by a Ward [Larson 1992] or Lafortune [Lafortune *et al.* 1997] model. Ashikhmin [Ashikhmin *et al.* 2000] for example produces good results for velvet by incorporating a special shadowing term.

Kautz and McCool [1999] approximate the four-dimensional BRDF by a product of two two-dimensional functions splitting viewing and light direction, which are stored as textures and combined during the rendering step. McCool et al. [2001] improved the above method by employing homomorphic factorization, leading to approximations with user controllable quality features. The above approaches were further improved [Ramamoorthi & Hanrahan 2002; Sloan *et al.* 2002; Latta & Kolb 2002], which all enable the BRDF to be lit by image based illumination while relying on different approximation functions. Unfortunately, their representations cannot easily be applied for realtime rendering of spatially varying materials.

BTFs

BTFs were introduced by Dana et al. [1999b]. A planar surface sample is lit by a directional light source and photographed from different directions. Thus the resulting images are a function of viewing and illumination direction, hence capturing effects caused by the mesostructure of a surface, like roughness, self-shadowing, occlusion, inter-reflections, subsurface scattering and color bleeding. Registering the different images of the BTF the data can be considered as a 6 dimensional reflectance field

$$L = L(x, y, \theta_i, \phi_i, \theta_o, \phi_o)$$

which connects for each surface point (x, y) of a flat sample the outgoing to the incoming radiance in the direction (θ_o, ϕ_o), (θ_i, ϕ_i) respectively. The measurement is done in RGB space, wavelength changes and time dependent effects like fluorescence are ignored.

Due to the computational complexity of the 6 dimensional function only a few realtime rendering algorithms exist [Kautz & McCool 2000; McAllister *et al.* 2002]. To achieve interactive rates, Kautz et al. use an approximation to an anisotropic version of the Blinn-Phong model and to the Banks model. In his recent work McAllister et. al. represented the 6D-reflectance field as a spatially varying BRDF. At each discretized surface position a Lafortune model is fitted and the parameters are stored in a texture map, which is called SBRDF. This representation can efficiently be evaluated in current graphics hardware. In addition to point and directional light sources their algorithm also supports image based illumination [Blinn & Newell 1976; Miller & Hoffman 1984; Greene 1986; Debevec 1998]. Though their algorithm yields good results for materials with low depth range, it proves unsatisfactory for more structured materials with high depth, as even for a high number of lobes the Lafortune model is hardly capable of capturing the variation in the reflectance behavior caused by the mesostructure.

5.3 Acquisition

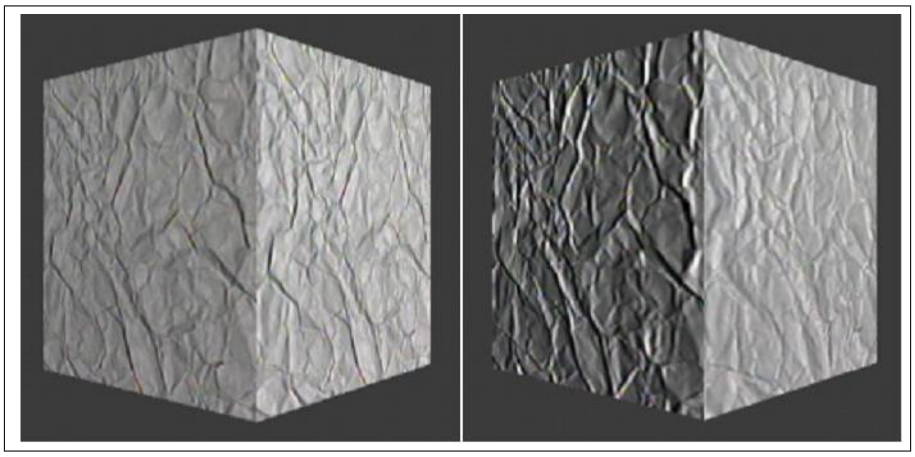


Fig. 5.3: The images show texture mapped cubes using the post-processed CUReT BTF data sample *crumpled paper*. In the left image only a frontal viewed texture is applied. The right image uses the complete BTF data set.

5.3.1 Measuring and synthesizing BTF data

In their pioneering work, Dana et. al. measured 61 samples of real-world surfaces and made them publicly available in the CUReT [Curet 2005] database. Unfortunately, the data is not spatially registered.

In order to demonstrate the enhancement over common texture mapping, a manually performed registration for a small number of samples and the mapping onto a cube is shown in Figure 5.3.

Self-shadowing and self-occlusion of the mesostructure on the surface are clearly visible. A drawback of the CUReT database is that it contains some graphical errors, caused by frame-grabber artifacts or reflections of the robot sample holder plate visible in the raw data.

Synthesizing BTF data addresses two problems. If only a discrete set of BTF samples is available it allows to synthesize the continuous BTF and furthermore it allows to synthesize BTF data of arbitrary size. Liu et al. [2001] registered some samples from the CUReT database using statistical properties and appearance preserving procedures.

Further methods to synthesize BTF data on a surface is described in Tong et al. [2002] using 3D textons or using histogram models [Dana *et al.* 1999a]. The advantages of these methods are the low memory requirements and that the overall structure and appearance is preserved. On the other hand, by introducing statistical and random components these methods destroy certain mesostructures, hence changing the BTF significantly and are not suitable for all kinds of materials, see for example [Tong *et al.* 2002].

In order to preserve the mesostructure measured image data is used, which is sampled dense enough to not require any synthesis and nevertheless stored in a compact form in memory. Because of the tileability of the used fabrics the size of the measured probe is sufficient.

5.3.2 Setup and Data Acquisition

The setup is designed to conduct an automatic measurement of a BTF that also allows the automatic alignment and postprocessing of the captured data. Restriction is made to planar samples with the maximum size of 10×10 cm. In spite of these restrictions measurement of a lot of different material types, for example fabrics, wallpapers, tiles and even car interior materials is possible.

As shown in Figure 5.4, the used laboratory consists of a HMI (Hydrargyrum Medium Arc Length Iodide) bulb (broncolor F575), a robot (intelitek SCORBOT-ER4u) holding the sample and a rail-mounted CCD camera (Kodak DCS 760). Table 5.1 shows two different samplings H_1 and H_2 of the halfspace of point X above the sample. According to the varying reflection properties of each sample, the sampling must be sparser or denser. A maximum of $n = 81$ unique directions for camera and light position is used resulting in an approximately equal sampling of the hemisphere.

θ_1 [°]	$\Delta\phi$ [°]	θ_2 [°]	$\Delta\phi$ [°]	No. of images
0	$-$*	0	$-$*	1
17	60	15	60	6
34	30	30	30	12
51	20	45	20	18
68	18	60	18	20
85	15	75	15	24

Tab. 5.1: Two different sampling densities H_1 and H_2 of viewing and illumination angles of the BTF database. *= only one image taken at $\phi = 0°$.

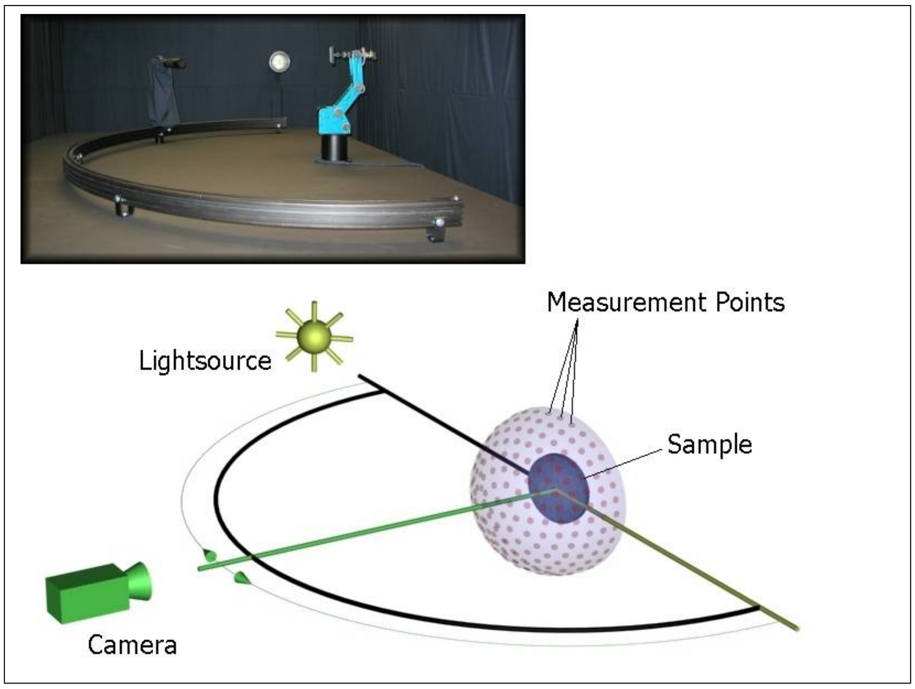


Fig. 5.4: Measurement setup consisting out of an HMI lamp, a CCD camera and a robot with a sample holder.

Figure 5.5 shows three measured samples: *CORDUROY*, *PROPOSTE* and *WOOL*. 6561 raw images were captured for each sample, each 6 megabytes in size (lossless compression) with a resolution of 3032×2008 pixels (Kodak DCR 12-bit RGB format). To ensure the correct correspondence of the measured reflection properties to a fixed surface position on the sample, close attention is paid to minimize positioning errors.

5.3.3 Postprocessing

After the measurement the raw image data is converted into a BTF representation, that is the perspectively distorted images must be registered. In this representation a complete set of discrete reflectance values for all measured light and viewing directions is assigned to each texel of a 2D texture. Registration is done by projecting all sample images onto the plane which is defined by the frontal view $(\theta = 0, \phi = 0)$.

To be able to conduct an automatic registration point and borderline markers are attached to the sample holder plate, as can be seen in Figure 5.6.

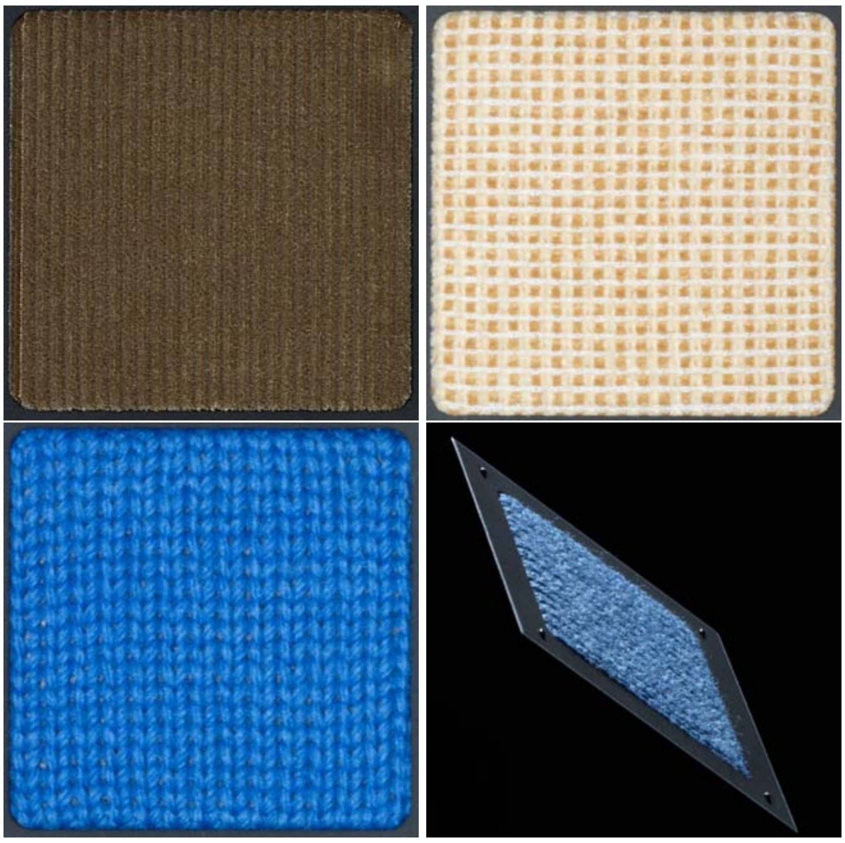


Fig. 5.5: Measured BTF samples; from left to right (top row): CORDUROY, PROPOSTE. Bottom row: WOOL frontal and perspective view.

After converting a copy of the raw data to black-and-white (8-bit TIFF),standard image processing tools are used to detect the markers during the measurement process. Restriction is made to the common 8-bit RGB texture format. To take advantage of the linear part of the camera response curve, the central 8-bit range of the 12-bit images is chosen. As a fixed focal length is used during one measurement, the maximum effective resolution of the sample holder in the image is 1100×1100 pixels. After all transformations are carried out, all images are rescaled to an equal size of 1024×1024 pixels, which now are called *normtextures* (N).

After this postprocessing step, the data amount of 167 gigabytes captured by the camera CCD chip is reduced to roughly 20 gigabytes of uncompressed data. By measuring planar probes of a certain size, the method relies on the tileablility of the fabrics. Therefore, a manually chosen region of interest (approximately

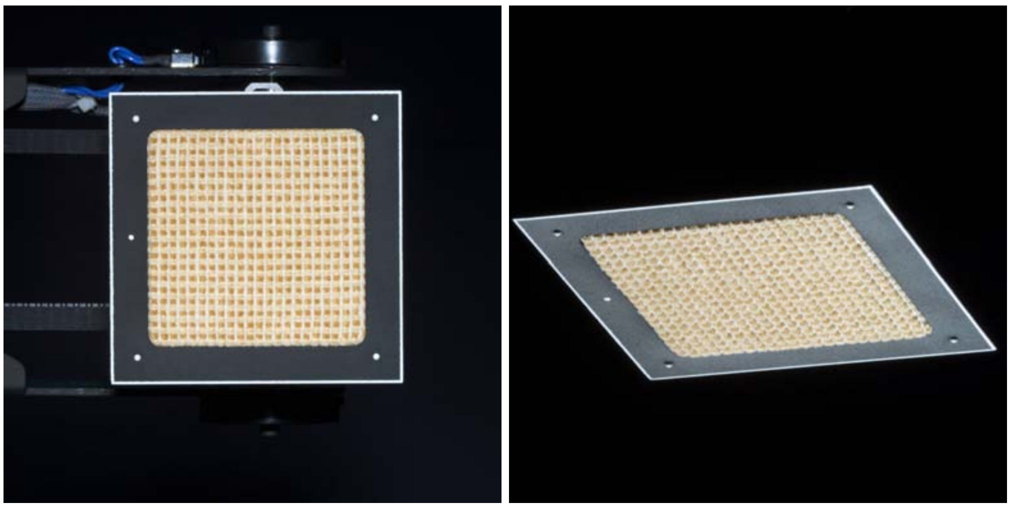


Fig. 5.6: Sample holder with the PROPOSTE sample. The left image shows the frontal view ($\theta = 0°$,$\phi = 0°$); the right image shows ($\theta = 60°$, $\phi = 342°$). White point and border markers are visible.

550×550 pixels) is cut out and resized. To create the final normtextures (256×256 pixels in size) linear edgeblending is applied, which reduces the usual tiling artifacts.

To allow a closer inspection of the measured fabrics, single point or directional light sources can be used. The easiest way to texture an object with a BTF texture would be to store a complete database in memory and fetch the nearest measured BTF image to the current viewing and lighting direction.

This way, the textures would approximately be viewed under the same angle they were acquired and therefore artifacts due to anisotropic sampling are avoided. The texturing can be done on a per face basis, introducing edge artifacts or on a per-vertex basis using blending, as described in [Chen *et al.* 2002].

Unfortunately the size of the database of one sample at a resolution of 256×256 pixels exceeds 1230 megabytes, which is not practical on today's hardware. To overcome this problem, the main idea is, to replace for each viewing direction the BTF defined by the normtextures into a series of basis textures by using a principal component analysis. Utilizing only a few components (≤ 16) of this series, the texture can be reconstructed at runtime.

5.4 Compression

Principal component analysis [Kendall 1975; Jolliffe 1986; Press *et al.* 1992] has been widely used to compress image data [Nishino *et al.* 2001]. Ramamoorthi [2002] showed by an analytic PCA construction, that using about five components is sufficient to reconstruct lighting variability in images of a lambertian object.

The measured samples all have a certain three-dimensional mesostructure, which leads to significantly varying surface appearance for changing viewing directions. To ensure a pixel position coherence, thus coping with the varying height of a surface position on the sample, a principal component analysis for each of the n viewing directions is done separately.

These directions are called *view slots* $S_j, j \in (1 \ldots n)$. Thus, in these slots the viewing direction is fixed so that only the light direction varies and therefore the analysis is done only on the effects caused by the changing illumination.

The n normtextures $N_{ij}, i \in (1 \ldots n)$ per view slot j are represented as vectors $X_{ij} = (r_{1,1}, g_{1,1}, b_{1,1}, \ldots, r_{h,w}, g_{h,w}, b_{h,w})$ of dimension $3 \times h \times w$, where h and w are the height and width of the normtextures, respectively.

A PCA of these vectors is performed, resulting in a series of eigenvalues $\lambda_{1j}, \ldots, \lambda_{nj}$ and eigenvectors $E_{1j}, \ldots, E_{nj}$ which corresponds to eigennormtextures $B_{1j}, \ldots, B_{nj}$ for this slot. The first $c < n$ eigennormtextures approximate any of the original normtextures N_{ij} in such a way that the sum of the squares of the projection errors onto the affine subspace spanned by $\{B_{1j}, \ldots, B_{cj}\}$ is minimized

$$N_{ij} \approx \sum_{k=1}^{c} p_{ikj} B_{kj}, \quad i = 1 \ldots n. \tag{5.1}$$

The coefficients $p_{ikj} = N_{ij} \cdot B_{kj}$ are weights, were $\cdot$ denotes the standard scalar product in $\mathbb{R}^{3 \times h \times w}$.

Figure 5.7 gives examples for reconstructed textures with a different number of eigennormtextures, and also shows difference images. For that, difference images are calculated using the length of the 8-bit RGB error vector between the original normtexture and the reconstructed images. Green color indicates a length of zero, whereas red indicates a length of 255 units.

Figure 5.8 shows the absolute eigenvalues for all components of three different view slots. The decay of the absolute values indicate the statistical dimensionality

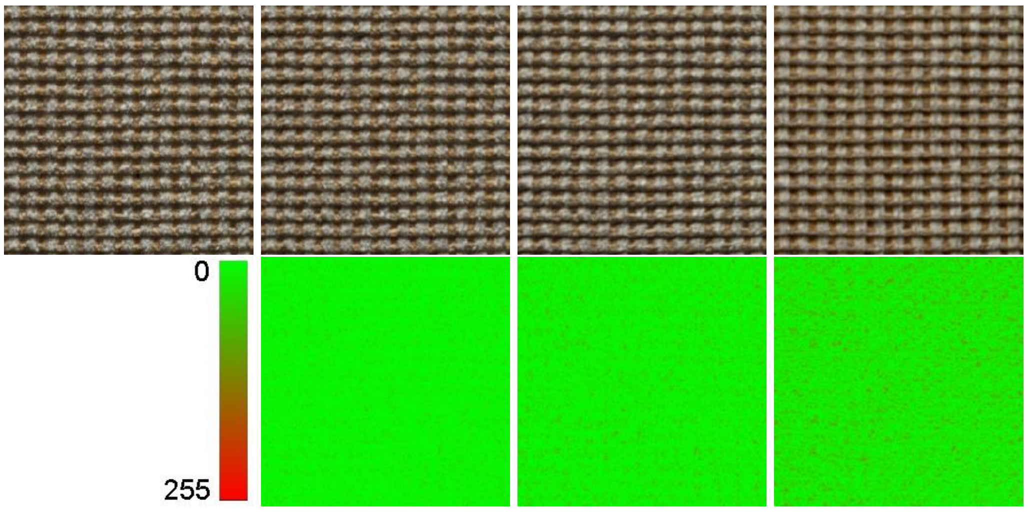


Fig. 5.7: Texture reconstruction using PCA. From left to right (top row): original normtexture, 16, 10, 5 components. Bottom row: difference images to the original.

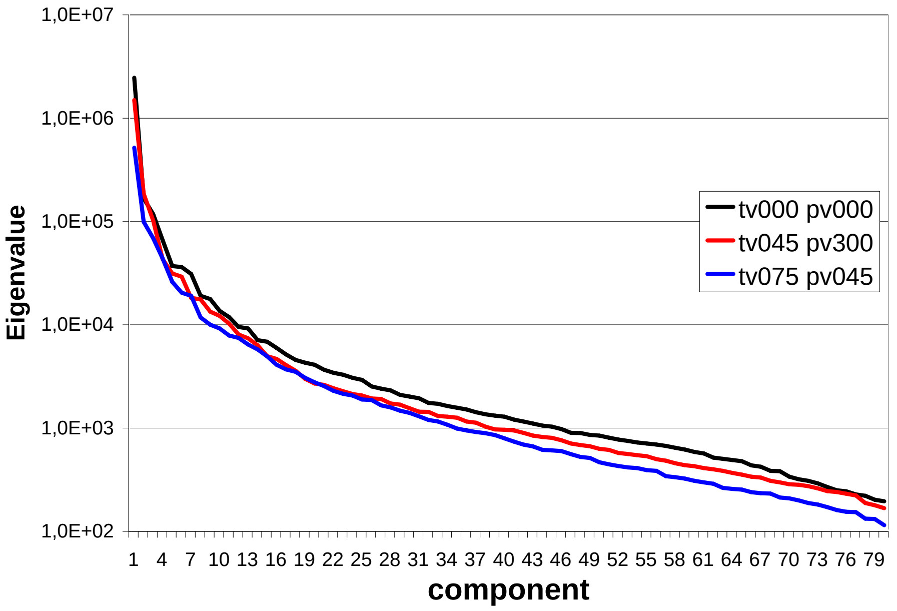


Fig. 5.8: Eigenvalues for the PROPOSTE sample in three different view slots j ($\theta = 0°$, $\phi = 0°$), ($\theta = 45°$, $\phi = 300°$), ($\theta = 75°$, $\phi = 45°$).

of the given normtextures. As the eigenvalues decrease rapidly in all the examples, $c = 16$ components were sufficient to reproduce the *look and feel* of the sample materials.

Note, that performing a principle component on the different view-slots reduces the size of the data set from about 1230 megabytes to 260 megabytes per sample for a 256×256 resolution.

5.5 Visualization

In this section the algorithm to reconstruct the texture T for a vertex V of a given triangle mesh at runtime is described. This is done while using a single point or directional light source. The emitted radiance $\mathbf{g}$ from the light source is stored as a three-component RGB float vector.

First the light and view vectors ($\hat{l}$, $\hat{v}$) for the vertex V are computed. Because of the memory requirements for storing the raw normtextures, now the representation of the textures as a series of basis normtextures B_{kj} is used. Choosing the nearest slot j corresponding to $\hat{v}$ and the weights p_{ikj} corresponding to $\hat{l}$ the texture T_j can be reconstructed.

$$T_j \approx \mathbf{g}\sum_{k=1}^{c} p_{ikj}B_{kj} \tag{5.2}$$

Because in general $\hat{l}$ does not match a measured direction exactly the known samplings H_1 or H_2 from the measurement are used to compute the four nearest measured light directions $i_m, m \in (1\ldots4)$ from the texture database for bilinear interpolation with the interpolation weights τ_m and with $N_{i_m j}$ denoting the reconstructed textures corresponding to i_m:

$$\begin{aligned}
T_j &\approx \mathbf{g}\left(\tau_1 N_{i_1 j} + \tau_2 N_{i_2 j} + \tau_3 N_{i_3 j} + \tau_4 N_{i_4 j}\right) \\
&= \mathbf{g}\sum_{m=1}^{4} \tau_m N_{i_m j} \\
&= \mathbf{g}\sum_{m=1}^{4} \tau_m \sum_{k=1}^{c} p_{i_m kj}B_{kj} \\
&= \mathbf{g}\sum_{k=1}^{c}\left(\sum_{m=1}^{4} \tau_m p_{i_m kj}\right)B_{kj} \\
&= \mathbf{g}\sum_{k=1}^{c} \gamma_{kj}B_{kj} \qquad (5.3)
\end{aligned}$$

This means, that the texture T_j is simply a weighted sum of basis textures.

$$T_j = \mathbf{g}\cdot\left(\gamma_{0j}B_{0j} + \gamma_{1j}B_{1j} + \ldots + \gamma_{cj}B_{cj}\right) \tag{5.4}$$

A *fragment program* is used to accomplish the reconstruction of the texture with $c = 16$ components using a ATI Radeon 9700. When blending the three resulting textures per triangle, a smooth transition is ensured as in [Chen *et al.* 2002].

If also view interpolation is desired, denote the four nearest view slots as $j_m, m \in (1 \dots 4)$ with the corresponding interpolation weights ω_m. Following (5.3) leads to:

$$\begin{aligned} T &= \omega_1 T_{j_1} + \omega_2 T_{j_2} + \omega_3 T_{j_3} + \omega_4 T_{j_4} \\ &= \omega_1 \sum_{k=1}^{c_1} \gamma_{kj_1} B_{kj_1} + \omega_2 \sum_{k=1}^{c_2} \gamma_{kj_2} B_{kj_2} \\ &+ \omega_3 \sum_{k=1}^{c_3} \gamma_{kj_3} B_{kj_3} + \omega_4 \sum_{k=1}^{c_4} \gamma_{kj_4} B_{kj_4} \end{aligned} \tag{5.5}$$

Note, that in the case of $j_1 \neq j_2 \neq j_3 \neq j_4$ four different eigennormtexture sets B_{kj_m} are needed.

5.5.1 Shadow Enhancements

In the context of cloth rendering, incorporating shadows and geometry self-shadowing is crucial for realistic rendering. Using point and directional light sources implies rendering hard shadow boundaries.

An efficient method for this purpose are the well known shadow maps [Williams 1978]. The calculation is hardware accelerated, for example through several OpenGL extensions [OSS 2005]. Nevertheless, a common problem with shadow mapping is projection aliasing [Akenine-Möller & Haines 2002]. Increasing depth buffer size and precision, as well as polygon offsets [Kilgard 2002] reduce these artifacts. Further improvements could be made using perspective shadow maps as introduced by Stamminger et. al [2002].

Unfortunately, in spite of self-shadowing, these artifacts are still visible, and destroy the realistic appearance of cloth. Therefore, the following method uses volumetric shadows, as proposed by Crow [1977]. The shadow volumes technique is artifact free [DeLoura 2001] and was enhanced to use hardware accelerated stencil buffers by Heidmann [1991]. A problem of the algorithm, when the camera is in a shadow volume, was solved by Bilodeau et. al and Carmack [1999; 2001]. Robust computation and hardware acceleration is also possible [Everitt & Kilgard 2002]. The computational complex shadow volume calculation can also be done in vertex shaders [Hart *et al.* 2001; Brennan 2002; Brabec & Seidel 2003], which minimizes one of the major drawbacks of this technique.

Nevertheless, there is a further problem, that leads to disturbing artifacts in cloth rendering: the shadow boundary always coincide with the silhouette of the mesh as seen from the light source. This silhouette is defined by those edges in the mesh which are incident to one front-facing and one back-facing triangle with respect to the light source position, respectively, see Figure 5.9 left side.

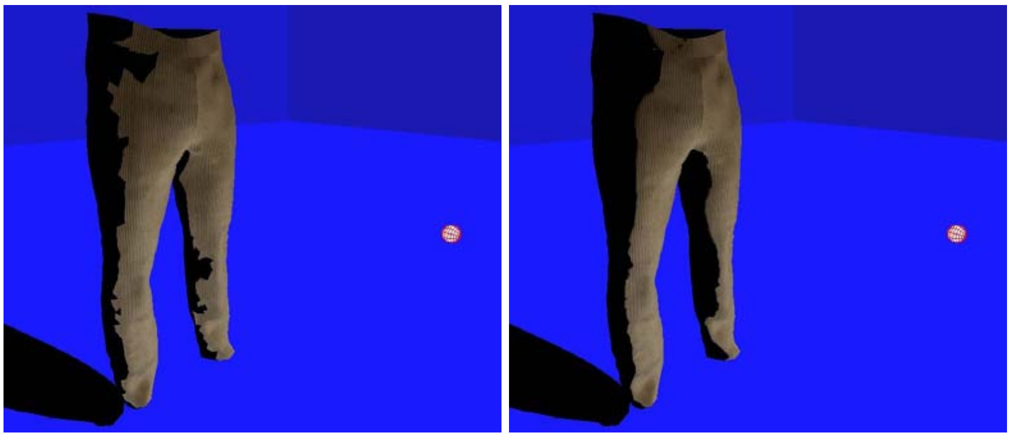


Fig. 5.9: Shadow boundaries. Left image shows the zigzag behaviour, which is gone in the right image using the described technique.

Therefore, in an arbitrary triangle mesh the silhouette edges does not define a smooth path but instead show a zigzag pattern. Note, that this is independent of the accuracy of the shadow computation and is worse for low-resolution meshes which are common in cloth modeling. Furthermore, if the light source moves the silhouette edge jumps between adjacent triangles leading to disturbing artifacts.

One way to cope with these problems is to consider the mesh as a smooth surface. This is actually also assumed during rendering, when interpolating the vertex normal vectors for lighting calculations. Using this observation leads to a simple solution to the problem.

If the sign of the scalar product between the normalized vertex normal $\hat{n}_1$ and the normalized light-vector $\hat{l}_1$ of V_1 and $\hat{n}_2, \hat{l}_2$, respectively, changes along an edge V_1V_2 of a triangle , the shadow boundary lies between these two vertices and the position of this boundary (P) on an assumed smooth surface can be estimated by the proportion of the angles at the two vertices V_1 and V_2, as shown in Figure 5.10.

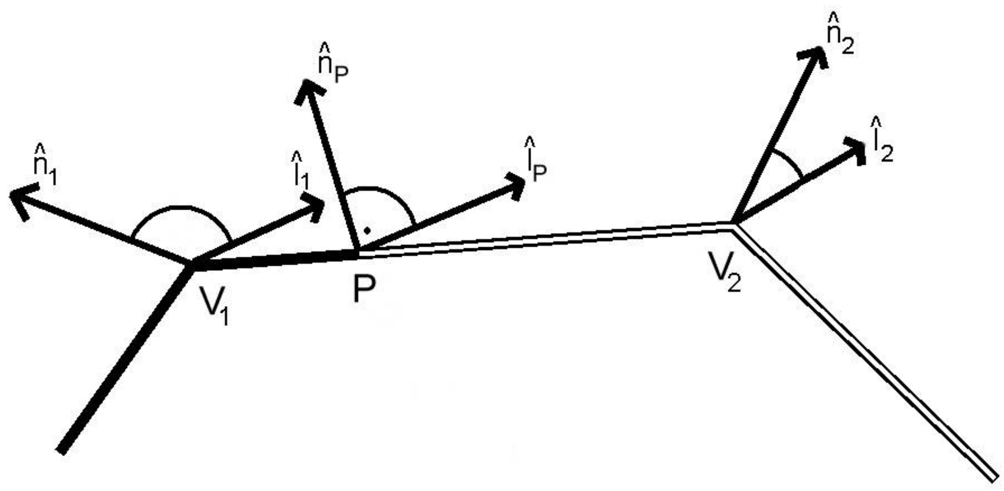


Fig. 5.10: Computing position P of the shadow boundary between V_1 and V_2.

Therefore, for each vertex V of a triangle one-dimensional texture coordinates are computed

$$(u)_V = (1.0 + cos(\alpha + \angle(\hat{n}_V, \hat{l}_V)))/2.0, \tag{5.6}$$

into a one-dimensional 1D half black and half white texture of size 1024 pixels. Using this texture leads to the smooth shadow boundary. In order to generate soft boundaries this texture can be blurred.

α is an offset, to compensate the popping artifacts, caused by the silhouette edge jumps, which was chosen to $\alpha = 15^\circ$ for the high resolution pair of trousers. The combination of this simple texturing method with the shadow volume algorithm delivers nice results as shown in figure 5.9 right side.

It should be noted, that the presented algorithm is also well suited for rendering dynamic meshes with macroscopic shadows, since the needed computations and the shadow volumes can be carried out each frame easily.

5.5.2 Image-based illumination

In this section, the algorithm is enhanced to illuminate the geometry using high dynamic range environment maps [Debevec & Malik 1997; Debevec 1998].

Next, the numerical integration of the rendering equation, the computation of visibility maps, the runtime algorithm for using image based illumination and a method to decompose the illumination of the geometry, to allow a faster change of the environment map are described.

Numerical integration of the Rendering Equation

Following the notation of Jensen [2001] at a surface point x the outgoing radiance L_o is given by

$$L_o(x, \mathbf{w}) = L_e(x, \mathbf{w}) + \int_S f_r(x, x' \!\rightarrow x, \mathbf{w}) L_i(x' \!\rightarrow x) V(x, x') G(x, x') dA', \quad (5.7)$$

where $\mathbf{w}$ is the outgoing direction, L_e the emitted radiance, S the hemisphere domain over x, f_r the BRDF, x' another surface point, L_i the incident radiance, $V(x, x')$ the visibility between the two surface points and $G(x, x')$ a geometrical term defined as

$$G(x, x') = \frac{(\vec{\mathbf{w}} \cdot \vec{\mathbf{n}})(\vec{\mathbf{w}'} \cdot \vec{\mathbf{n}'})}{\|x' - x\|^2} \quad (5.8)$$

with the normal $\mathbf{n}$ at x and $\mathbf{n}'$ at x', respectively. The proposed method sets the emitted radiance $L_e(x, \mathbf{w}) = 0$ and does not compute any inter-reflections. The hemisphere domain is discritized using a hemicube [Cohen & Greenberg 1985], which leads to

$$L_o(x, \mathbf{w}) \approx \sum_{\alpha} f_r(x, p_\alpha \!\rightarrow x, \mathbf{w}) L_i(p_\alpha \!\rightarrow x) V(x, p_\alpha) G(x, p_\alpha), \quad (5.9)$$

where p_α is a pixel of the hemicube.

Visibility Map Pre-computation

Because image based illumination is used and the radiance values are stored in an environment map, a lookup into this map has to be provided. Therefore, *visibility maps M* are precalculated for each vertex. These maps store a discretization of the hemisphere of the vertex V, which is a hemicube with its top side perpendicular to the vertex normal.

Figure 5.11 (left image) shows an unfolded hemicube. Using a color-coded environment map (center image) a look-up table into a high dynamic range map (right

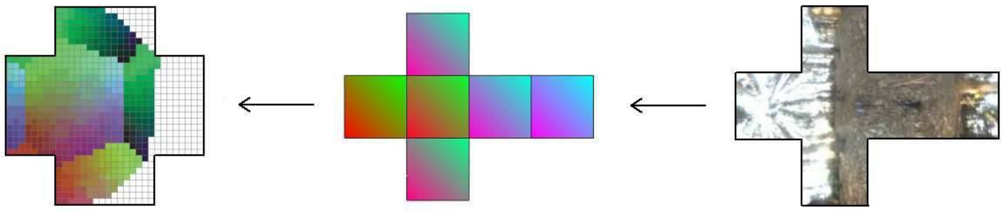


Fig. 5.11: Visibility map computation. Visibility map (left) with rendered color-coded lookup environment map (middle). White color in the visibility map stands for occlusion caused by the mesh. On the right side a HDR environment is shown, which is mapped onto the color-coded one.

image) is created. This allows easy exchange of the environment map. By also rendering the geometry itself, macroscopic self-shadowing is included.
Because a pixel p_α represents a certain direction $(V \rightarrow p_\alpha)$ and does not necessarily match one of the measured directions, the visibility map is subdivided into n direction patterns, as seen in the Figure 5.12.

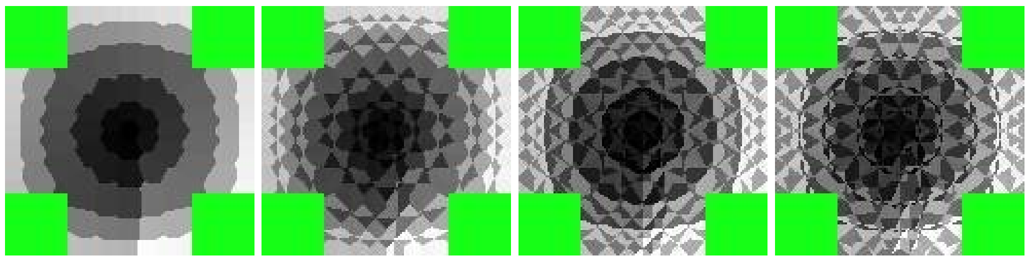


Fig. 5.12: Visibility map (64×64 resolution) direction encoding with n grey levels. From left to right: Nearest, second nearest, third and fourth nearest direction.

The four nearest measured directions in respect to $(V \rightarrow p_\alpha)$ are assigned to p_α. This allows for a bilinear interpolation with the interpolation weights $h_{d_k}, k \in (1 \ldots 4)$ for all four directions d_k. A visibility map pixel now stores the following information:

- visibility of a pixel of the environment map and if it is visible, the position of this pixel in the map
- four nearest measured directions in respect to the direction represented by this pixel
- corresponding interpolation weights

Real-time Algorithm using image-based Illumination

It would require $n = 81$ multi-texturing passes for each triangle to incorporate all measured light directions, which cannot be done in real-time. In the following, the usage of the visibility maps and the representation as a series of basis normtextures, to illuminate a triangle mesh using high dynamic range images is described.

First the view vector $\hat{v}$ for each vertex V is calculated and the nearest view slot j is chosen. At this point the radiance $\mathbf{g_i}$ coming out of the n measured directions at each vertex has to be evaluated. Similar to equation 5.2 the texture T_j now is computed as follows:

$$\begin{aligned} T_j &\approx \sum_{i=1}^{n} \mathbf{g_i} N_{ij} \\ &= \sum_{i=1}^{n} \mathbf{g_i} \sum_{k=1}^{c} p_{ikj} B_{kj} \\ &= \sum_{k=1}^{c} \left(\sum_{i=1}^{n} \mathbf{g_i} p_{ikj} \right) B_{kj} \\ &= \sum_{k=1}^{c} \gamma^*_{\mathbf{kj}} B_{kj} \end{aligned} \tag{5.10}$$

Introducing a multiplication factor f, denoting the exposure level of the high dynamic range map, the texture T_j is reconstructed very similar to 5.4:

$$T_j = f \cdot (\gamma^*_{\mathbf{0j}} B_{0j} + \gamma^*_{\mathbf{1j}} B_{1j} + \ldots + \gamma^*_{\mathbf{cj}} B_{cj}) \tag{5.11}$$

It should be noted, that now $\gamma^*_{\mathbf{kj}}$ is also a three component float vector.

In order to compute $\mathbf{g_i}$ for a vertex V, for all $p_\alpha \in M_V$ a lookup into the environment map at the position stored in p_α is performed. The radiance $\mathbf{r}$ stored at that position is assigned to $\mathbf{g_{d_k}}$ and weighted with h_{d_k}. Here $d_k, k \in (1 \ldots 4)$ denotes the four directions stored with p_α as described above.

For view interpolation the same calculations as in (5.5) have to be applied. $\gamma^*_{\mathbf{kj}}$ is computed for all vertices $V_\zeta, \zeta \in (1 \ldots N)$ were N is the number of vertices of the geometry. Thereby a new vector U is introduced, which stores all $\gamma^*_{\mathbf{kj}\zeta}$:

$$U = (\gamma^*_{111} \cdots \gamma^*_{c11}, \ldots, \gamma^*_{1nN} \cdots \gamma^*_{cnN}) \tag{5.12}$$

with the dimension $3 \times c \times n \times N$.

This vector has do be calculated once per environment map and allows the real-time change of the viewing position and of the exposure f.

A drawback of this method is, that changing the environment map implies a complete new calculation of $\mathrm{g_i}$ for all vertices. This heavily depends on the visibility map resolution and the number of vertices and therefore on the hardware rendering speed.

Reducing the visibility map resolution adaptively to achieve interactive changing rates introduces under-sampling artifacts of the environment map during motion, which can be compensated if the change stops, by using an adaptively higher resolution for the visibility map.

Decomposition of the Environment Map

To overcome the problems mentioned in the last section, now a new decomposition method is proposed. Again principal component analysis is used.

As aforementioned, all incoming radiance have to be evaluated, if the object is rotated or the environment is changed. Daily observation shows that for example rotation of an object under natural illumination leads only to slight irradiance changes on the object surface, if the rotation angle is small.

The key idea is, that a set of vectors $U_a, a \in (1...A)$ is computed, where A denotes the number of different environment maps used. Performing a PCA on these vectors, results in a series of new eigenvalues and eigenvectors . The latter correspond to eigenweightsets $W_1, \ldots, W_A$. The first $e < A$ eigenweightsets can be used to approximate any of the original weightsets U_a:

$$U_a \approx \sum_{k=1}^{e} o_{ak} W_k, \quad a = 1 \ldots A. \tag{5.13}$$

The coefficients $o_{ak} = U_a \cdot W_k$ are weights, were $\cdot$ denotes the standard scalar product in $\mathbb{R}^{3 \times c \times n \times N}$.

To test the method, an object is rotated relative to the environment and the vector U is computed for each rotation step. This is equal to the usage of several different environment maps. By using $\nu = 12°$ degree steps $A = 30$ weight sets are obtained. Furthermore, a high resolution environment and visibility map (256×256 pixels) are used. A comparison between reconstructed ($e = 5$ eigenvectors) and original images is shown in Figure 5.13.

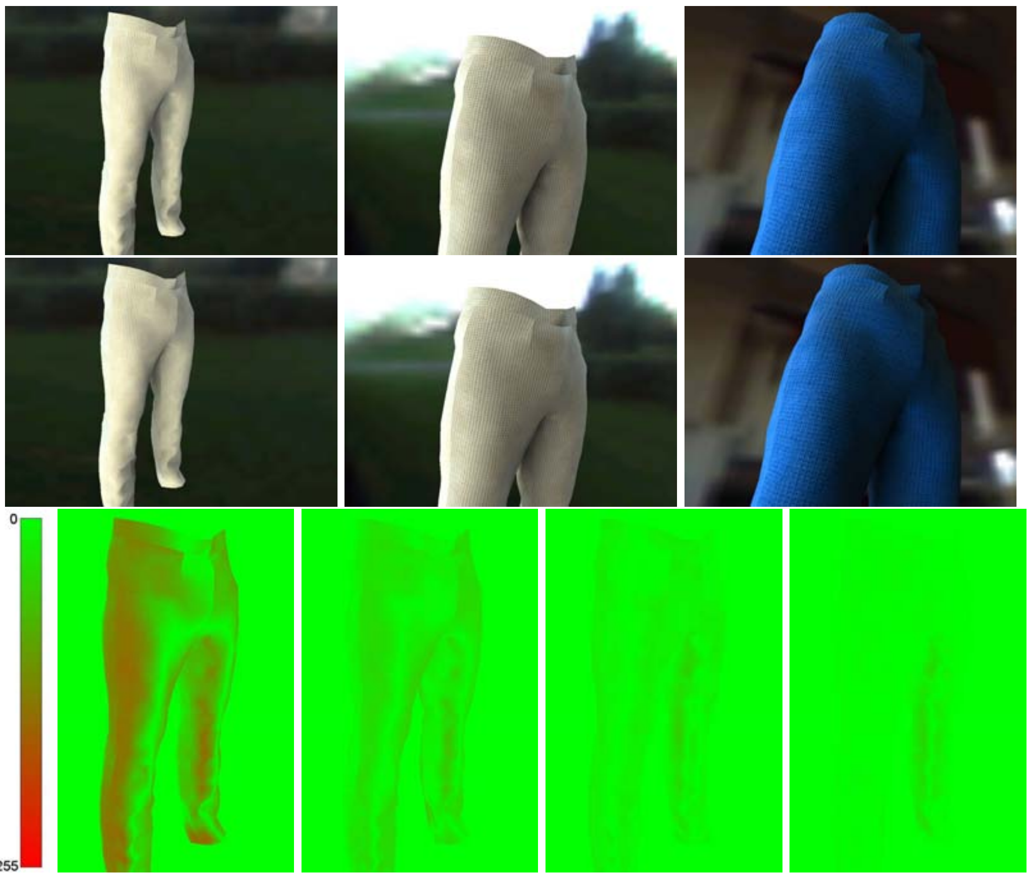


Fig. 5.13: Decomposition of the illumination of the geometry (from second to bottom row) original, reconstructed and difference error images. In the error image green denotes no error, while red denotes maximum error, see text for details. From left to right: 2,3,4 and 5 PCA components were used. The reconstruction was done with $e = 5$ eigenweight sets.

Here, the length of the RGB error vector between the original and the reconstructed images is calculated. Green color indicates a length of zero, whereas red indicates a length of 255 units. Increasing the number e of the used eigenweightsets clearly minimizes the error.

Figure 5.14 shows the weights o_{ak} for all $A = 30$ sets for all eigenvectors. It should be noted, that the oscillation is denoting the rotation around the object axis. As a result, the object or the environment now can be rotated, hence the lighting situation and the view can be changed at interactive frame rates. The

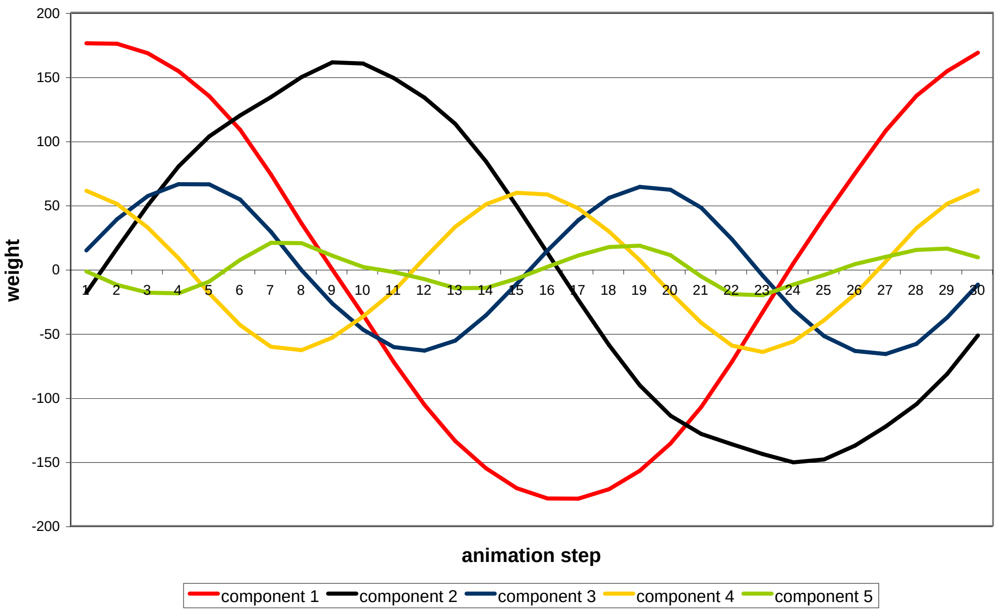


Fig. 5.14: Change of the weights for each component during the animation. Note the oscillating, denoting the rotation of the object.

complete weight set U for a desired rotation angle $\delta \in (0 \ldots 360°)$ is reconstructed at runtime.

5.6 Visualization Results

All results were obtained under Windows 2000 on a 1.5GHz Athlon with a ATI Radeon 9700 graphics accelerator and OpenGL. For the texture reconstruction a *fragment program* (GL_ARB_fragment_program) and multi-texturing are used. The Eigentextures for all samples have a resolution of 256×256 and need about 260 megabytes memory per sample.

Video VIII

Figure 5.2 and all images in Figure 5.24 and 5.17 were rendered at a 1280×960 resolution. The used mesh complexity range from nearly 800 vertices to 9200 vertices. More result images are shown in Figure 5.16. Additional animations are shown in the video.

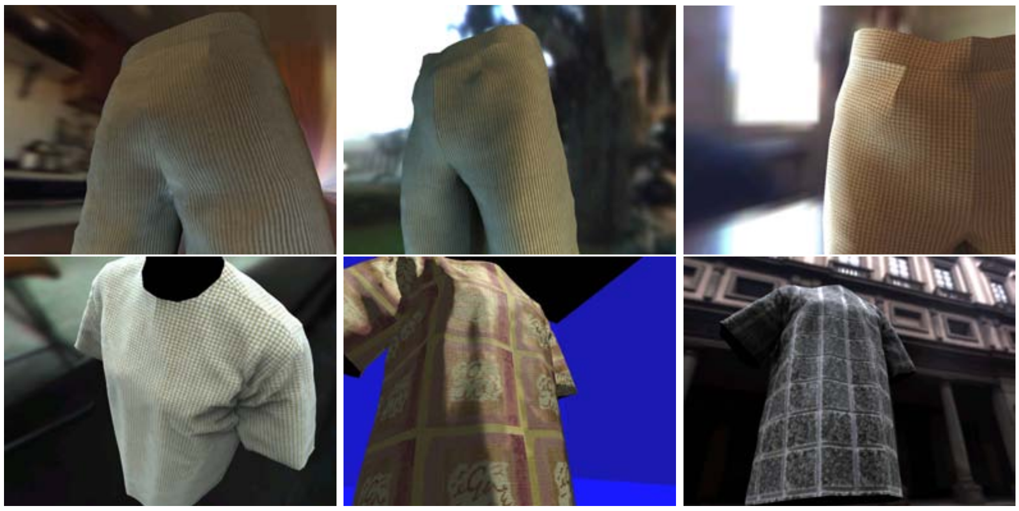


Fig. 5.15: Result images. Top row (from left to right): CORDUROY sample in Kitchen and RNL environment, PROPOSTE in Kitchen; next row: PROPOSTE in Building, WALLPAPER with point light source and STONE in Uffizi.

The algorithm is capable of rendering several BTF sets onto one geometry, if proper texture coordinates and material id's per vertex are supplied and sufficient system memory is available (see also the avatar in figure 5.17). The algorithm handles non-fabric materials (*WALLPAPER, STONE*) as well, as can be seen in figure 5.24 in the second row.

Table 5.2 gives an overview over the frame rates achieved. While enabling the decomposition, 2.0 frames per second are achieved, including the dynamic change of illumination and camera position, instead of the recalculation time of 3700 milliseconds per change for the shirt mesh (900 vertices).

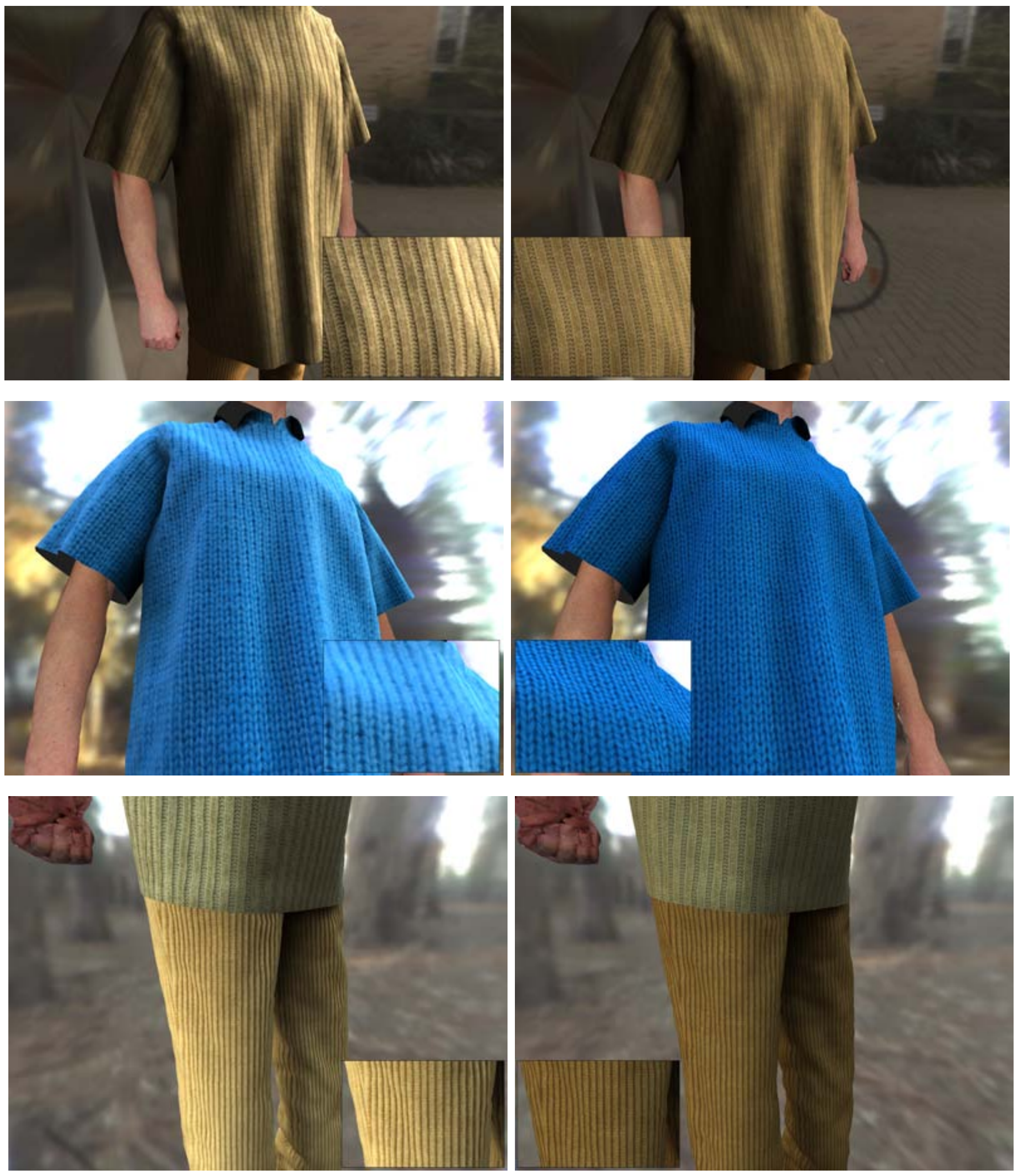


Fig. 5.16: Effects of using BTF data of different materials under natural illumination conditions. The images on the left side show the BTF rendering, while the images on the right side were rendered using only normal texturing (frontal texture only). The mesostructure and the characteristic reflection properties are completely missing, the Look & Feel is gone. This can be seen especially in the close-ups. As high-dynamic range backgrounds some data from [Debevec 1998] is used.

To conclude, the presented method is able to capture and visualize reflection pro-

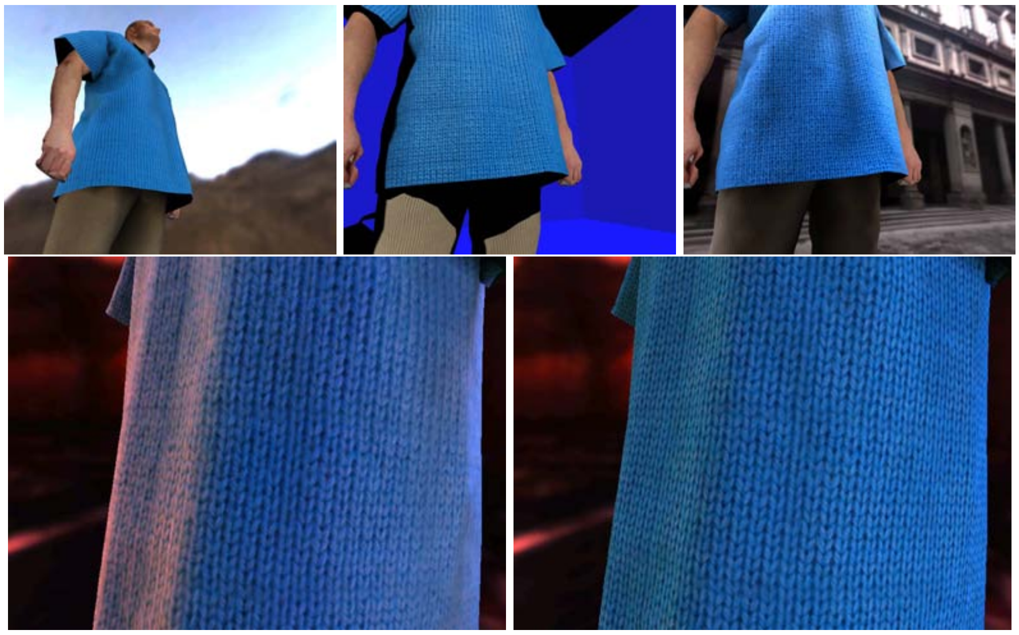


Fig. 5.17: Top row (from left to right): WOOL and CORDUROY with avatar at Beach, with point light source and in Uffizi. Next row: WOOL sample in Grace environment, left using BTF data, right normal texturing. Notice the angular illumination dependence.

mesh name	vertices	illumination method	average frame rate [FPS]	HEM update time [msec]
shirt	900	PLS	9.3	
shirt	900	HEM	9.5	3.8k
shirt	9208	PLS	1.3	
shirt	9208	HEM	1.1	38.5k
pair of trousers	833	PLS	9.5	
pair of trousers	833	HEM	10.1	3.7k
pair of trousers	5222	PLS	2.1	
pair of trousers	5222	HEM	2.1	23.2k

Tab. 5.2: Results for different meshes and illumination models. The frame rates were obtained using four times view blending with a total of 16 PCA components. PLS=point light source, HEM=high dynamic environment map.

perties of cloth at interactive frame rates. The approach decouples reflection properties from geometry while preserving the *look and feel* of a fabric, including important mesostructural features.

Furthermore, the image based illumination allows the usage in a desired clothing shop environment. With the presented decomposition method, interactive change of viewing and illumination is possible. While using single point or directional light sources a simple but effective method to compute smooth shadow boundaries was added.

With the emersion of new graphic hardware in the near future, which supports more multi-texturing operations per pass the four times view blending could be done in one pass and/or the number of used PCA components could be increased.

5.7 Silhouette Enhancements

For the realistic visual impression of an object, not only the em look & feel of the rendered material is important. Besides the usage of measured material reflection properties, the correct rendering of the silhouette parts of an object, giving important visual clues, has to be considered. This section proposes a method to use measured material silhouette properties with textured, low resolution meshes, to visually enhance the objects silhouette with effects like fur, lints or the thread of a tyre.

Normal or bump maps simulate a more detailed geometry, but can not cope with these effects. The algorithm uses measured bidirectional texture functions (BTF), a height field representation of the material sample, precomputed visibility information and incorporates illumination by high dynamic environment maps and point or directional light sources.

All previous approaches do not take special care of the silhouette regions of the objects. But the silhouette gives important visual clues [Koenderink 1984], and has to be considered especially when using low polygonal model and texturing. Viewed under certain illuminations, correct rendering of for example fur or lints has to be taken into account. And if the used texture suggests a macrostructure to the viewer, which is the case, for example for tread or coarse wool, the objects silhouette has to be changed accordingly.

5.7.1 Previous Work

The *fins and shells* rendering technique was introduced by Lengyel et al. [2001] to render fur in real-time. While this approach was also texture based, the authors used only one randomly chosen texture from a geometric hair model. Isidoro et al. [2002] enhanced this technique by incorporating a light model, which is also used in [Heidrich & Seidel 1999]. Both methods do not use image based illumination nor do they use measured data. Other approaches for fur and hair rendering are based on volumetric data [Kajiya & Kay 1989; Van Gelder & Wilhelms 1997].

As already explained in Section 2.1.8, the missing normal information of simplified meshes needed for correct shading can be stored easily in normal maps, which can be shaded efficiently in software or on programmable graphics hardware. As with bump maps [Blinn 1978], a detailed geometry is only simulated via shading and perturbed pixel normals. As stated above, with low polygon objects, missing macroscopic silhouettes at the objects silhouette edge are a clearly visible problem.

To detect an object´s silhouette edges, several methods exist, based on normal comparison or probabilistic approaches [Markosian *et al.* 1997]. The common method is also used with volumetric shadows, as proposed by Crow [1977]. Robust computation and hardware acceleration is possible and can also be done in vertex shaders [Brennan 2002; Hart *et al.* 2001; Brabec & Seidel 2003]. In the following, the silhouette edges are computed with the latter technique.

Other papers dealt already with the problem of the enhancement of object silhouettes. Sander et. al. [2000] stored the edge information from a highly detailed model into an hierarchy and used stencil buffer techniques to apply this information to a coarse mesh, resulting in a smooth silhouette. They also introduced an edge antialiasing algorithm [Sander *et al.* 2001] to smooth polygonal silhouettes.

Previous work in cloth rendering falls into two main categories. The first is the explicit modeling of the underlying mesostructure and rendering it using volumetric techniques, introduced by Kajiya et al. [1989], storing opacity values and other reflection properties.

While certain approaches are not real-time capable [Gröller *et al.* 1995; Gröller *et al.* 1996; Xu *et al.* 2001], some interactive methods exist which use special shading models [Daubert *et al.* 2001; Daubert & Seidel 2002]. All volumetric approaches depend on their used voxel resolution, wether they can provide sufficient macrostructure silhouettes or not. Rendering lints and other above surface effects were not addressed so far for silhouette edges. Recent texture based approaches [Dana *et al.* 1999b; McAllister *et al.* 2002; Sattler *et al.* 2003] lack also this effects.

With low polygon cloth models using a technique like [Sander *et al.* 2000] is not possible, because a high resolution mesh does not exist. The following method extends the texture based approach of Sattler et al. [2003], to be able to handle also object silhouettes correctly. The method is a combination of BTF and *fin & shell* rendering.

5.7.2 Acquisition & Postprocessing

This section describes the acquisition of BTF fins (BTFF). A more detailed measurement setup can be found in Sattler et al. [Sattler *et al.* 2003].

θ_L [°]	$\Delta\phi_L$ [°]	No. of images
0	$-^*$	1
15	60	6
30	30	12
45	20	18
60	18	20
75	15	24

Tab. 5.3: θ_L and ϕ_L values for the light directions used for the acquisition. 81 images were taken. *= only one image taken at $\phi_L = 0°$

The acquisition itself is done automatically, using a robot arm and a fixed HMI (Hydrargyrum Medium Arc Length Iodide) light source (broncolor F575), which emits a sun-like spectrum. 81 different light directions θ_L, ϕ_L (see Table 5.3) were measured, using a fixed view of $\theta_V = 90°$, $\phi_V = 0°$ on the upper edge of the sample. The images were captured against a black background to easily obtain the alpha mask for each image. Colored backgrounds proved not to be good enough to extract the fine fur details. Figure 5.18 shows the robot arm holding the wool sample and an extracted image for $\theta_L = 45°$, $\phi_L = 200°$. The sample is spanned over a wooden sample holder to have a half cylinder like form. More details of the setup can be found in [Sattler *et al.* 2003].

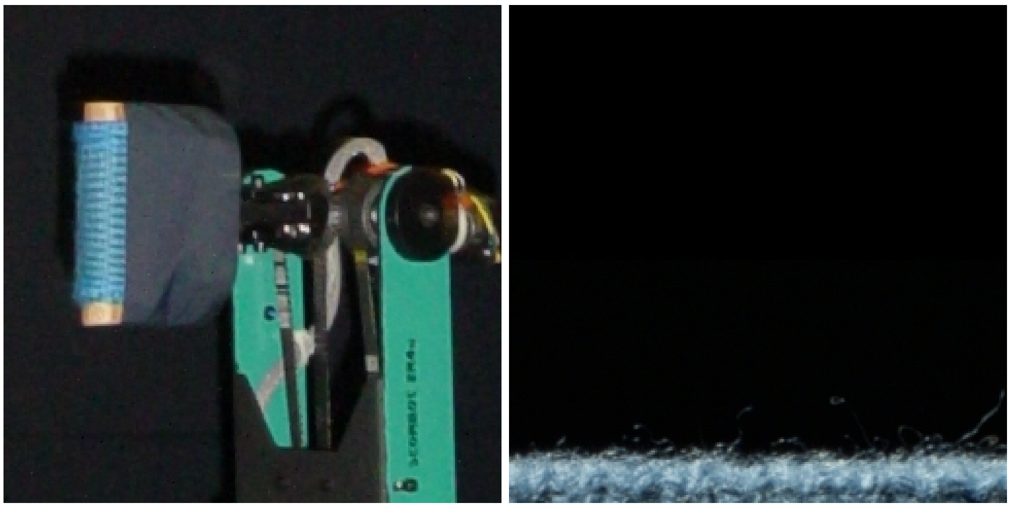


Fig. 5.18: Robot arm holding the WOOL sample and an extracted image for $\theta_L = 45°$, $\phi_L = 200°$

Note, that depending on the complexity of the material fin structure, more measurements might be necessary. Figure 5.19 gives an overview, what should be measured. S is the material sample, R the region of interest, cut out later. S is subdivided into G_i regions, where i depends on the material complexity. ϕ_V is the rotation angle of the material sample on the sample holder and G_i the measuring slices, viewed with $\theta_V = 90°$. For the WOOL sample one fixed view is sufficient to produce good results.

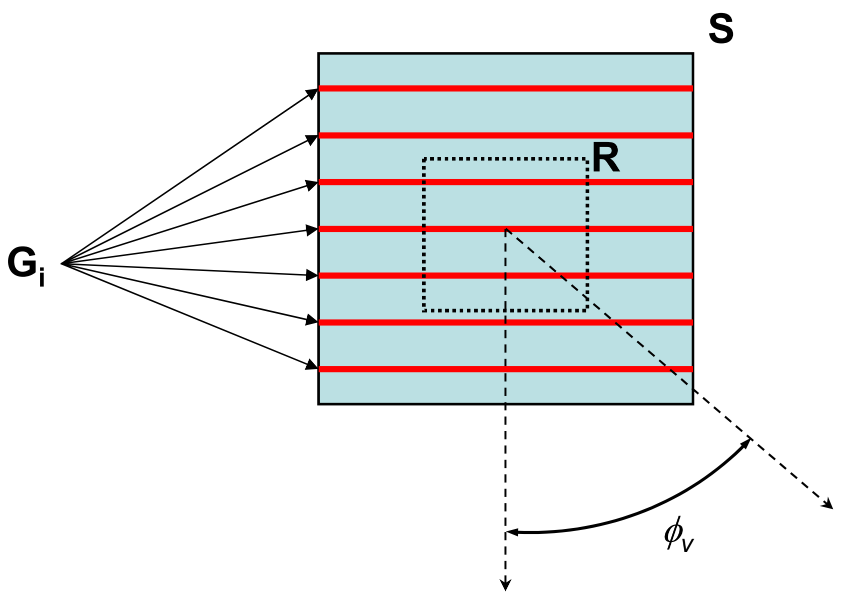


Fig. 5.19: Sketch of the measurement needed for materials with complex structured surfaces.

All measured 81 images were registered, a region of interest is chosen and square result F images were cut out. The extraction of the background color is done entirely on the GPU, as explained later.

Similar to the basis method in Sattler et al. [2003], a principal component analysis is used, to reduce the data amount.

The first $c < n$ eigennormtextures approximate any of the original normtextures F_{ij} in such a way that the sum of the squares of the projection errors onto the affine subspace spanned by $\{B_1, \ldots, B_c\}$ is minimized

$$F_i \approx \sum_{k=1}^{c} p_{ik} B_k, \quad i = 1 \ldots n. \tag{5.14}$$

The coefficients $p_{ik} = F_i \cdot B_k$ are weights, where $\cdot$ denotes the standard scalar product in $\mathbb{R}^{3\times h\times w}$.

5.7.3 Rendering

In this section the basic algorithm for rendering BTFF is described. First of all, all silhouette edges for a given viewpoint are computed at runtime. As stated, this can be done hardware accelerated, defining a set of edges E_F which are called *fin edges*. Each edge $E \in E_F$ is defined by two vertices V_1, V_2.

As described by Lengyel [2001] vertex normals are used to compute the two needed fin vertices V_3, V_4. Using point or directional light sources the direction of the incoming radiance at V_1, V_2 is computed. Then, a fin is reconstructed as follows using the eigentextures B_k:

$$F_i \approx \mathrm{g}\sum_{k=1}^{c} p_{ik}B_k \tag{5.15}$$

where g is the float vector holding the radiance for the light source. Because there are two vertices with different incoming radiances a fading between the two fin ends is computed. This is done by using two blend textures $blend01$ and $blend02$ which simply are linear black white color interpolations. All color channels (.r.g.b.) are added up into the alpha channel (.x) of the image. After this, the noise induced by the camera is compensated and both blend results are added together. The following fragment program (5.1) accomplishes this task (shown for the left part only $result01$. Here, $alpha\_adjust$ compensates any camera noise and is chosen manually to 0.04.

```
!!ARBfp1.0
MUL result01, blend01, result01;
MOV result01.w, result01.x;
ADD result01.w, result01.w, result01.y;
ADD result01.w, result01.w, result01.z;
SGE temp.x, result01.w, alpha_adjust;
MUL result01.w, temp.x, result01.w;
ADD endresult, result01, result02;
```

Listing 5.1: ´fragment program for fin blending

The Figure 5.20 shows a reconstructed WOOL fin with the alpha channel generated by the fragment program. The fin on the right side shows the original measured image. The left image shows the reconstructed fin rendered against a green background. To show the fading between two different radiance (RGB) triplets, the left outermost part of the reconstructed fin is lit by red light and the right part is lit by white light.

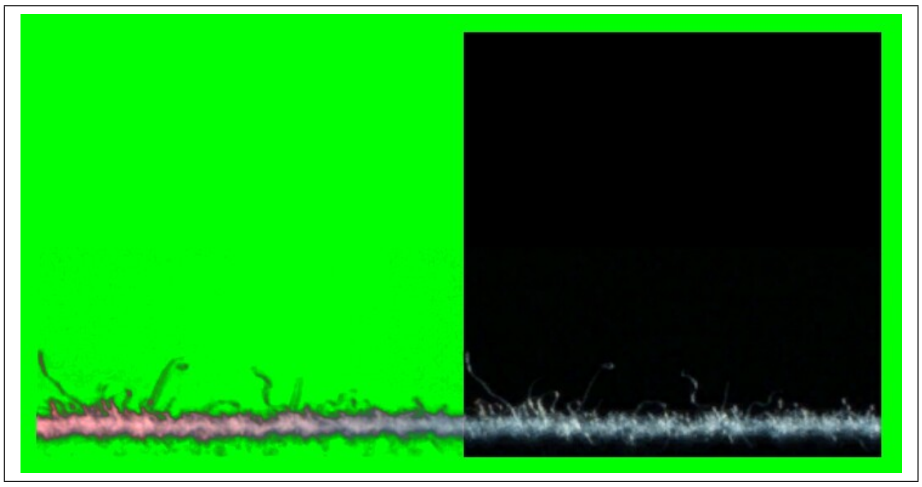


Fig. 5.20: Example of a reconstructed WOOL fin.

Texture Coordinate Computation

All fin textures are equal in size. While rendering a silhouette edge $E(V_1, V_2)$, defined by the vertices V_1 and V_2, texture coordinates have to be computed. The following calculations are done in texture space (S,T). Because only one fin set is used, first the distance D from the texture space origin to the first texture coordinate at V_1 is computed:

$$D = \sqrt{{S_{V_1}}^2 + {T_{V_1}}^2} \tag{5.16}$$

To do correct scaling of the fin texture, the length L of the edge in texture space is calculated:

$$L = \sqrt{(S_{V_1} - S_{V_2})^2 + (T_{V_1} - T_{V_2})^2} \tag{5.17}$$

With the material specific fin offset O, which defines the beginning of the fin space above the sample, the complete set of texture coordinates is computed:

$$V_1 : L, O \tag{5.18}$$
$$V_2 : D, O \tag{5.19}$$
$$V_3 : L, 1.0 \tag{5.20}$$
$$V_4 : D, 1.0 \tag{5.21}$$
$$\tag{5.22}$$

BTFF Rendering Problem

Due to the unknown surface curvature of the geometry, several silhouette edges can overlap each other with regard to the current viewing position. Without depth sorting the fins and drawing them back to front, alpha blending artifacts can occur, as can be seen in Figure 5.21 (left image). Because depth sorting all needed fins is computational expensive at runtime, in the following a simple 2-pass rendering is proposed as described with the following pseudo code (5.22) and the result shown the right image.

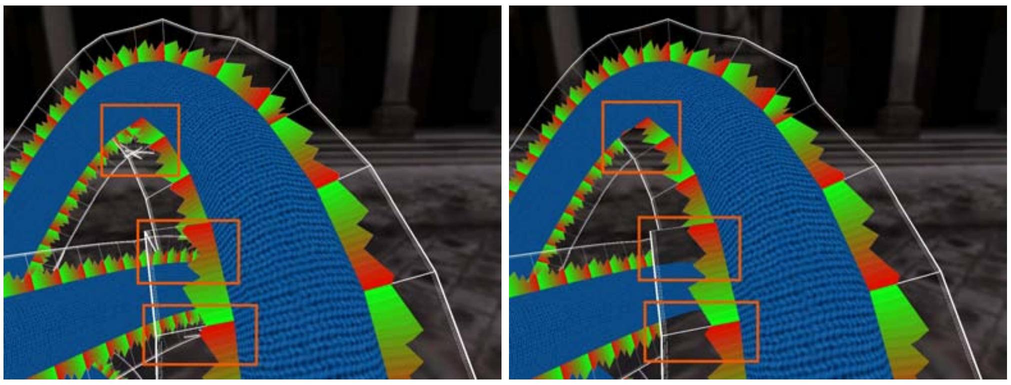


Fig. 5.21: BTFF alpha blending problem. In left image BTFF are drawn without sorting. In the right image, the problems are gone, using the described algorithm.

Image based illumination

In the case of image based illumination a visibility map [Sattler *et al.* 2003] is pre-computed for each vertex of the base geometry. The map is a texture, which stores a lookup into an high dynamic range environment map for 81 directions in the local vertex coordinate frame. This texture is later on used to weight the different

```
procedure DrawAllFins()
glDepthMask(GL_FALSE);
glDisable(GL_CULLFACE);
GenerateFinTexture()
glBindTexture(GL_TEXTURE_2D, fintexture);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_BLEND);
for all fins do
    DrawTheFins()
end for
glDepthMask(GL_TRUE);
for all fins do
    DrawTheFins()
end for
```

Fig. 5.22: Pseudo code to solve the BTFfin rendering problem.

PCA components for the rebuild of each fin. See also [Sattler *et al.* 2003] for a detailed description. Note, that without precomputation of the visibility maps, which is the case for point light sources, the BTFF can also be computed for dynamic objects.

5.7.4 Results & Conclusions

The method was implemented within an interactive hardware-accelerated OpenGL system. The results were obtained under Windows 2000 on a 1.5GHz Athlon with a ATI Radeon 9700 graphics accelerator. For the texture reconstruction a *fragment program* and multi-texturing is used. The Eigentextures for all samples have a resolution of 256×256 pixels.

The Figures 5.23 and 5.24 show the difference between BTFFin usage and normal BTF texturing (without fins). The preprocessing time for using BTFF is less than one minute. The storage requirements are fairly small, around 16 additional textures for the BTFF are required.

The presented method for cloth rendering clearly enhances the silhouette region. The visual appearance for low polygon models is improved and should be used if BTF rendering of the material is also applied. The needed data is easily captured and due to compression only a little memory overhead is added compared to

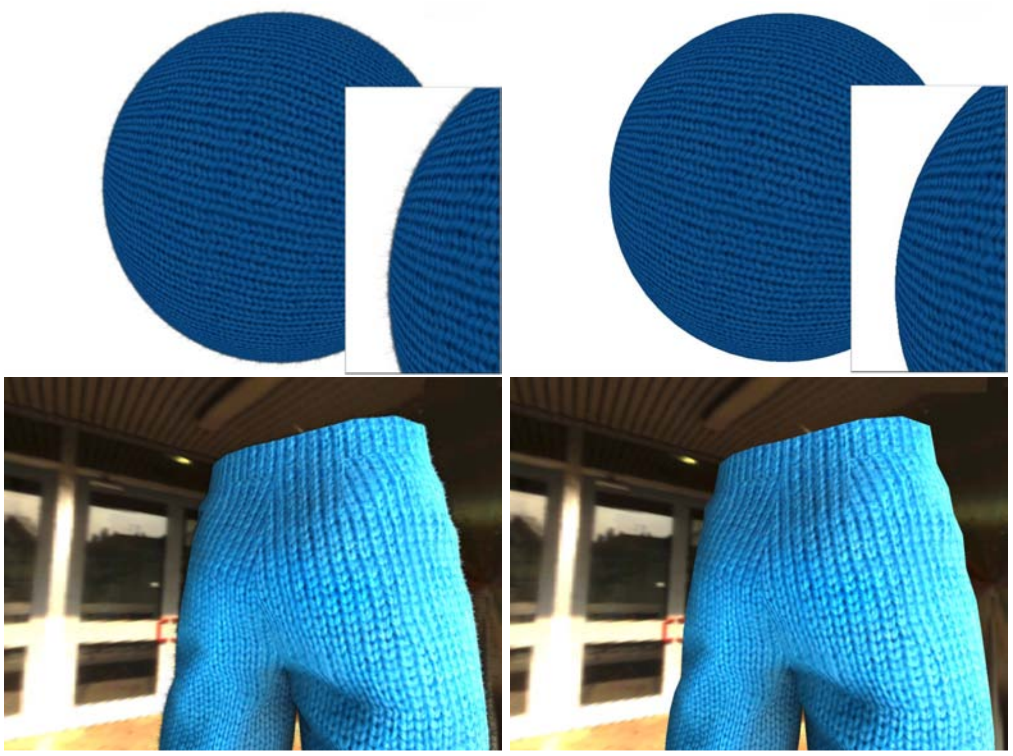


Fig. 5.23: Result images using a simple sphere object (top) and a complex pair of trousers (bottom row) with the WOOL data set. On the left side, BTFF are used, on the right side only conventional BTF rendering is applied.

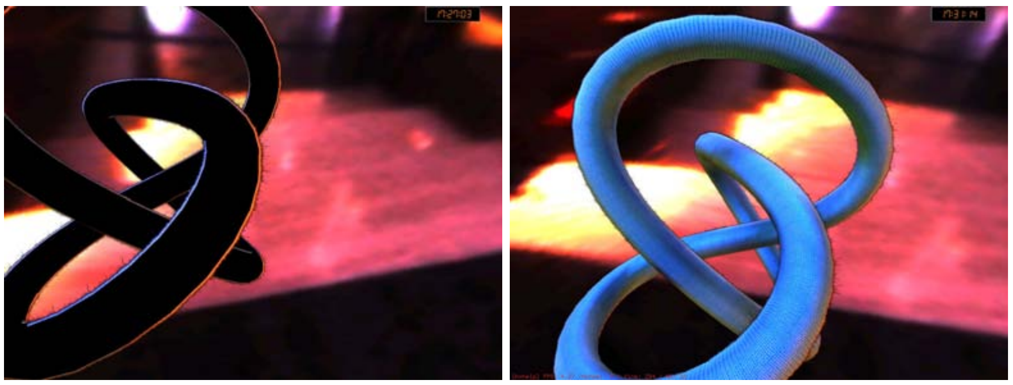
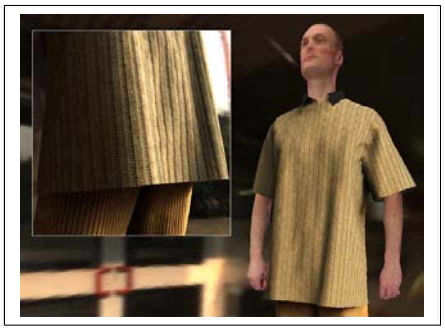


Fig. 5.24: Result images using objects within a complex lighting environment.

normal fin rendering.

CHAPTER 6 Applications

6.1 Introduction



This chapter introduces two industrial projects, which are directly correlated to the algorithms and methods described in the previous parts of this thesis.

The first project, *Virtual Try-On* (VTO), deals with cloth rendering. The process chain spans from customer virtualization over virtual tryon to bespoke clothing. Here, the focus lies on customer decision support.

The main topic of the second project, *RealReflect* (RR), is car and interior rendering. In contrast to VTO, the focus lies on manufacturer design support and visual quality of VR systems. The process chain includes acquisition, processing and the rendering pipeline of different types of materials and geometries.

6.2 Virtual Try-On

The *Virtual Try-On* [VTO 2005] project lasted from March 2001 until June 2004 and was funded by the German Ministry of Education and Science (BMBF) under project 01IRA01A. Table 6.1 lists the project partners.

partner	country
HUMAN SOLUTIONS GmbH	GER
University of Tübingen	GER
University of Bonn	GER
Fraunhofer Institute for Computer Graphics	GER
Hohenstein Institute for Clothing Physiology	GER
DFKI GmbH	GER
Odermark	GER
Cove & Co	GER

Tab. 6.1: All project partners for the VTO project.

6.2.1 Project Description

As stated in the introduction, VTO was designed to assist in customer decision for clothing. Figure 6.1 shows an overview over the different aspects of the project.

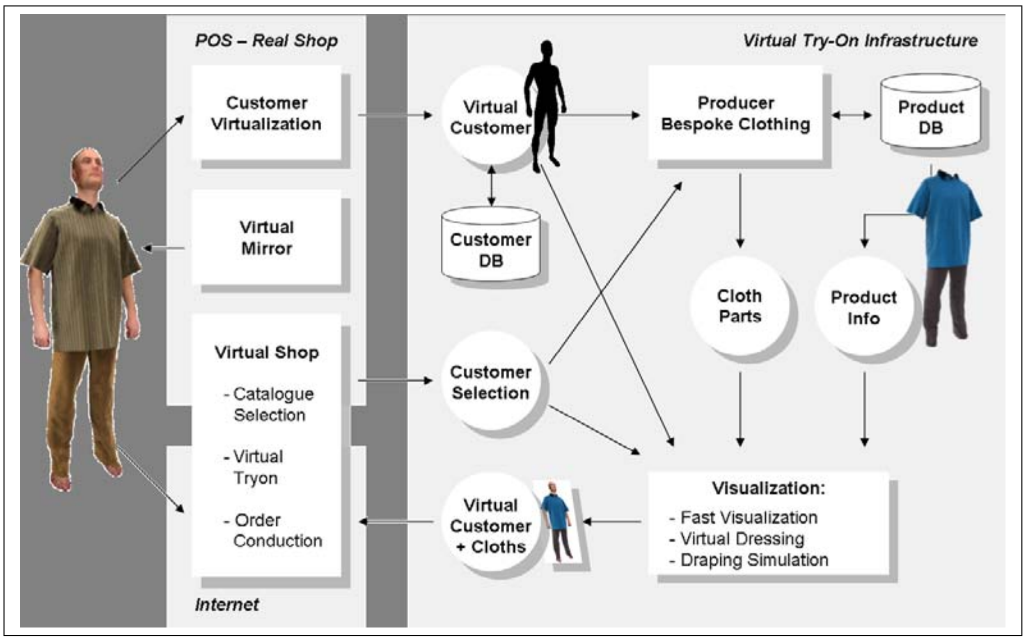


Fig. 6.1: Overview of the project pipeline.

The main idea of the project is to pre-visualize cloth on a real-world body of the customer before it is manufactured. This allows the customer to answer fitting and aesthetic questions in beforehand.

The concept is centered around a real-world point-of-sale (POS) and an e-commerce application. Within the POS, the customer body data can be scanned and after consultation with the sales person, the virtual cloth can be viewed on a virtual mirror. Here, different types of material and cloth can be visualized and the chance can be reduced, that the final cloth is returned due to unsatisfactory customers.

In the back-end of the system, cloth manufactures can maintain databases which contain materials, cloth parts and additional customer data, which is used to generate the final visualization.

In the following, several stages of the infrastructure are explained in detail.

Customer Virtualization

First, the customer has to be virtualized. That is, the body geometry has to be automatically measured and transferred into the computer, see Figure 6.2. This should be combined with textural information about the customers skin to ensure a high self-recognition in the subsequent stages.

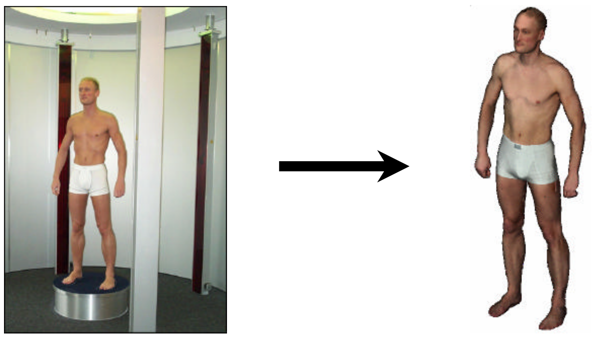


Fig. 6.2: Body scanning technology is used for customer virtualization.

Virtual Shop

As shown in Figure 6.1, a *Virtual Shop* is also included in the project, to support the e-commerce section of the project (see Figure 6.3). Here, in combination with a customer database and without the direct access to a real point-of-sale, orders can be carried out.

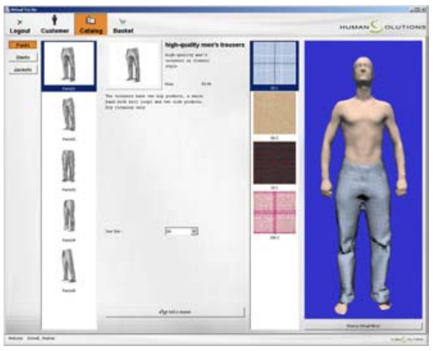


Fig. 6.3: Screenshot of the virtual shop.

Data Input

One essential part of the photo-realistic visualization is the usage of real-world optical material surface properties. Therefore, each available material has to be measured and stored for usage within the system, as shown in Figure 6.4. Further details about the measurement and compression can be found in Chapter 5.

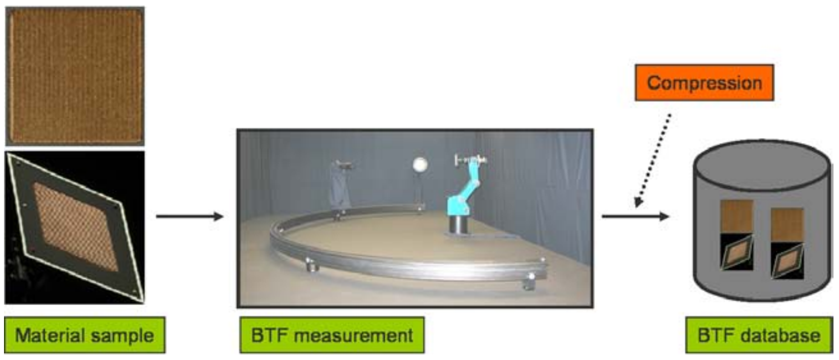


Fig. 6.4: Measurement and storage of the optical material properties.

Prepositioning

After the customer has chosen the material and cloth type, automatic prepositioning is necessary to ensure a smooth simulation of the cloth on the virtualized customer. This process, which is based on 2d cloth patterns, is shown in Figure 6.5.

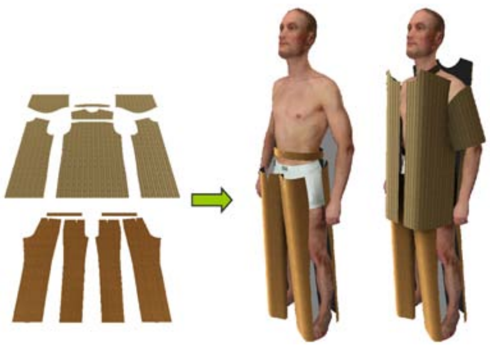


Fig. 6.5: Several prepositioning steps based on cloth parts.

Dressing & Draping Simulation

Besides the optical material properties, the mechanical properties were also measured. This allows for a complete cloth simulation, which lead to a realistic generation of cloth geometry. Several time steps of the simulation are shown in Figure 6.6.

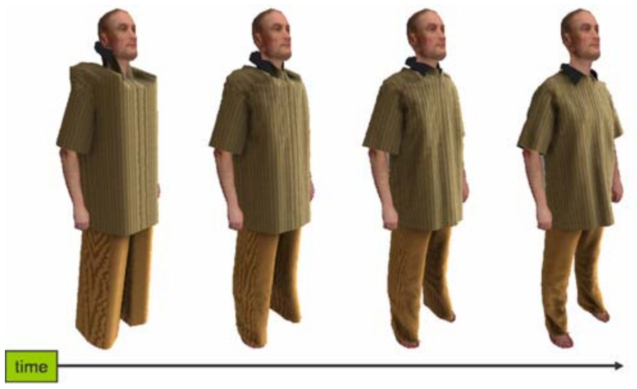


Fig. 6.6: Dressing & draping advances with time.

Virtual Mirror

Within the POS a *Virtual Mirror* (Figure 6.7) allows the customer to see a virtual counterpart, which finally supports the buying decision.

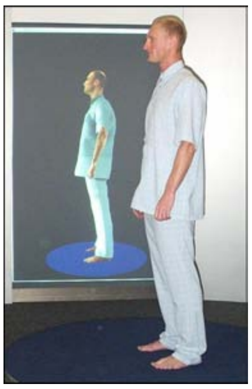


Fig. 6.7: Virtual Mirror with real-world person and clothing.

Final Rendering

For all output channels, that is the Virtual Mirror, the e-commerce applications or handouts, high-quality BTF rendering is used, which is also explained in Chapter 5.

In the following, several project-related result images are shown (6.8). There is also a video available on the thesis DVD. Further details about the project can be found in [Wacker *et al.* 2004; Wacker *et al.* 2005].

Video 6.2

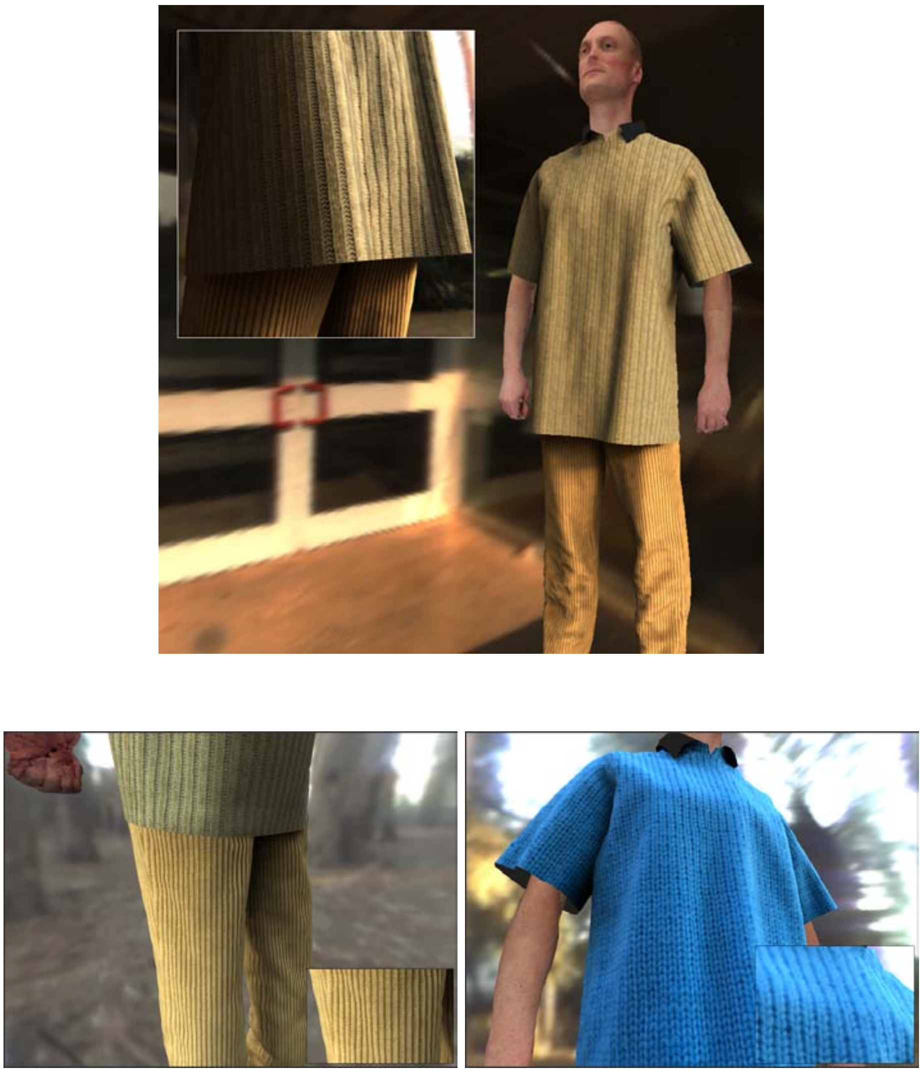


Fig. 6.8: Several project-related result images.

6.3 Real Reflect

The *RealReflect* [RealReflect 2006] project lasted from March 2002 until April 2005 and was funded by the European Union under project IST-2001-34744. All shown car models were kindly provided by the DaimlerChrysler AG. Table 6.2 lists the project partners.

partner	country
Vienna University of Technology	AUT
University of Bonn	GER
UTIA Prague	CZE
MPI for Computer Science	GER
INRIA Grenoble	FRA
DaimlerChrysler AG	GER
Faurecia	FRA
vr architects	AUT
IC:IDO GmbH	GER

Tab. 6.2: All project partners for the RR project.

6.3.1 Project Description

As stated in the introduction, RR was designed to assist in manufacturer design support for car interior and architecture. Figure 6.9 shows an overview over the input part of the project pipeline.

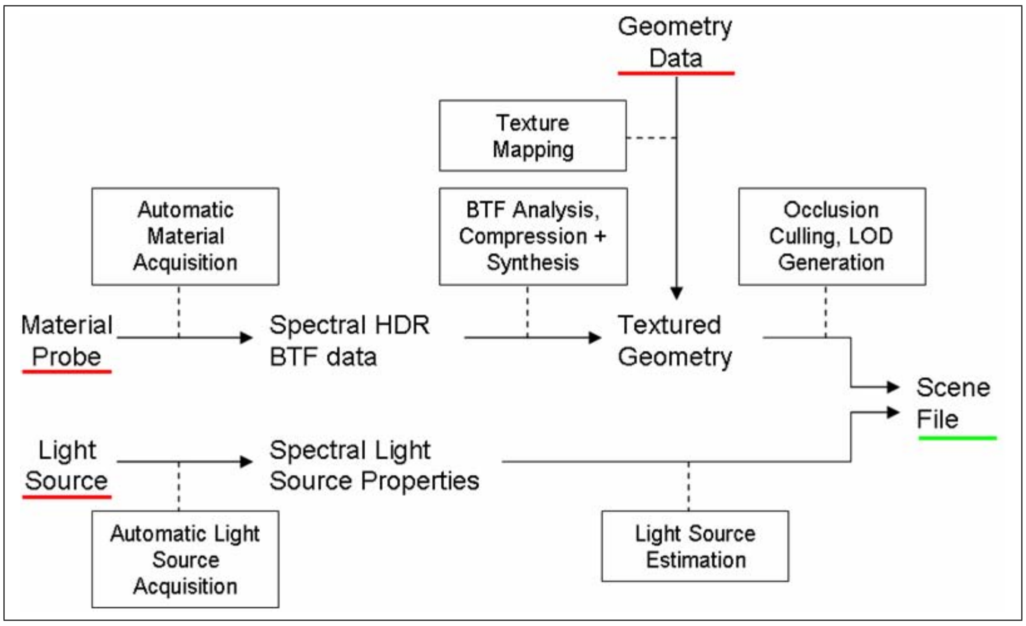

Geometry Data
Texture Mapping
Automatic Material Acquisition
BTF Analysis, Compression + Synthesis
Occlusion Culling, LOD Generation
Material Probe
Spectral HDR BTF data
Textured Geometry
Scene File
Light Source
Spectral Light Source Properties
Automatic Light Source Acquisition
Light Source Estimation

Fig. 6.9: Overview of the input part of the project pipeline.

The main idea of the project is to create a process chain for high-quality car interior and architectural design system, which in huge parts is fully automatic.

The usage of measured real-world data for materials, lighting and geometry allows for a degree of simulations, which is unalterable for safety design issues, for example car interior lighting. It allows also for rapid prototyping and supports also decision support for new design and manufacturing models. Besides the phase, which handles all the input data, the second rendering phase includes also tone mapping and integration of the projects parts into existing VR systems.

In the following, several stages of the preprocessing phase, which results in the *scene file*, are explained in detail.

Preprocessing Phase

As with the VTO project, real-world data is needed for input data (see Figure 6.10).

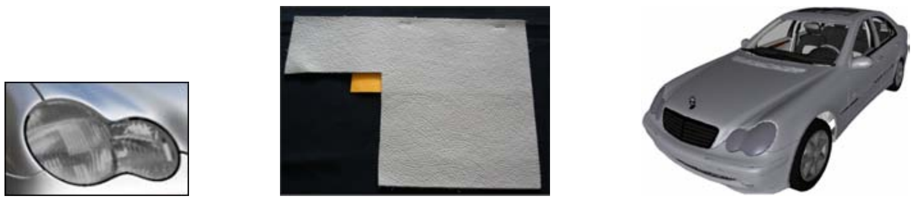


Fig. 6.10: Input details for the pipeline. Light source data, material probes and geometry data.

Here, *automatic material acquisition* of the optical material properties, *light source acquisition* and *real-world geometry* form the integral input parts. The measured data is further processed and later used for visualization.

Therefore, the gathered BTF data is analyzed and compressed. Automatic texture coordinate generation (see Figure 6.11) is performed and the properties of the used light sources are captured.

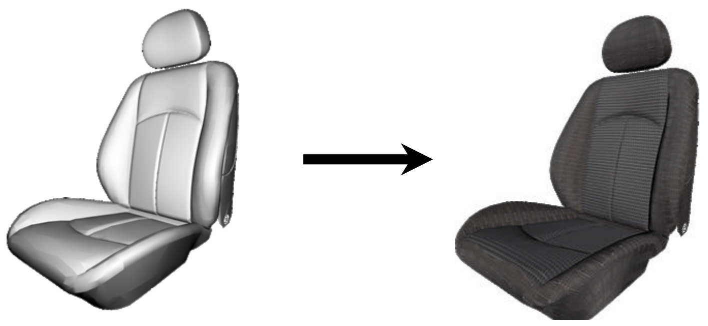


Fig. 6.11: Automatic texture coordinate generation.

For performance reasons, occlusion culling and LOD generation of the geometry files are performed. All processed information is stored in the *scene file*, which is transferred to the second rendering phase (see Figure 6.12).

Rendering Phase

The rendering phase uses the scene file as input and has two major rendering queues. The first is real-time BTF rendering, which is used for VR systems or systems which require interaction. The second takes advantage of a full global illumination simulation to produce high-quality stills.

The generated *temporary image* is further processed with *real-time tone mapping* to produce the final image.

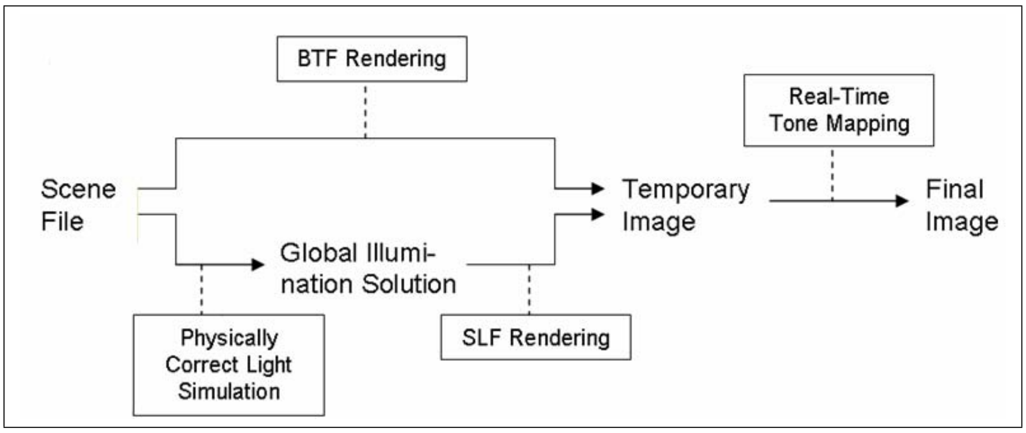


Fig. 6.12: Rendering part of the process pipeline.

Figure 6.13 shows several exemplary renderings.

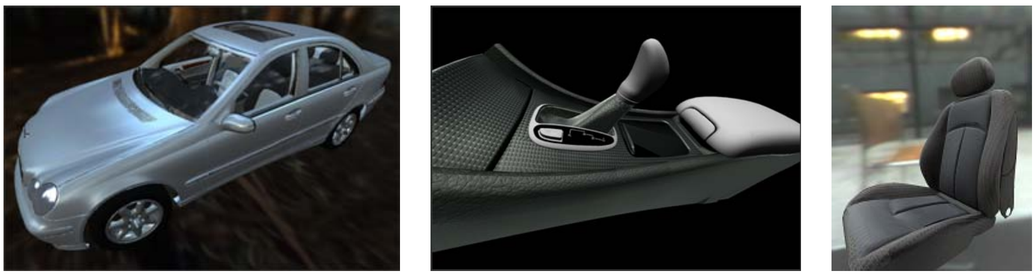


Fig. 6.13: Full car, seat and middle console final renderings.

CHAPTER 7 Conclusions & Future Work

In this chapter I will summarize the main contributions of this thesis and explain my thoughts for future research directions in the field of cloth visualization.

7.1 Conclusions

Recalling the first overview diagram in Figure 1.1, important aspects of a lot of topics in cloth visualization in computer graphics are addressed in this work.

Cloth Visualization

The main topic of this thesis is *cloth visualization*. Besides the following relevant subtopics, this work describes the first integrated acquisition and rendering pipeline using measured real-world material reflection properties (see Chapter 5).

The pipeline was successfully used in an industrial project (see Chapter 6) and includes also a ground-breaking compression scheme for the measured data.

Shadows

While the necessity for realistic shadows in real-world scene is obvious, the algorithms to efficiently compute these shadows are by far straight forward. In the context of cloth visualization especially self-shadowing is important. Chapter 4 introduces two new methods to efficiently compute these kind of shadows using graphics hardware acceleration. The first method is optimized to handle static geometry and is integrated into the BTF rendering pipeline. Evaluation of rendered hemi-cubes is used for visibility computations.
The second method provides a solution for animated meshes, for examples avatars with clothing. Here, visibility computation is performed using occlusion queries.
Shadow computation itself is closely related to perception. Understanding the human visual system therefore might provide hints to optimize shadow computation by leaving out perceptional less important parts. An experiment is described which uses level of detail optimized shadow calculation and perception evaluation to speed up the shadow calculation process.

Animation

Animation topics addressed in this thesis are motion sequence generation and sequence compression. Both are closely related to cloth visualization. Besides a method to easily and automatically create new animation sequences (3.2.1), also a state-of-the-art compression scheme for the animation geometry is presented (3.3.1), which is adopted from the compression scheme of the material reflection properties (see below).

Material Reflection Properties

Concerning optical material reflection properties, which play an essential role within cloth visualization, three main topics are addressed in this work: *acquisition*, *compression* and *rendering* (see Chapter 5). The measurement is done via a computer-controlled semiautomatic robot assembly and is designed to measure real-world material samples. The compression scheme uses principal component analysis (PCA) to analyze and sort the data for efficient compression without too much loss of visual quality. The rendering system uses hardware accelerated multi-texturing to decompress and render several materials under natural illumination in real-time.

7.2 Future Work

Is the topic *cloth visualization* in computer graphics finished yet? Is it depletive researched? Of course not.

I still believe, that the *holy grail* for all topics in computer graphics is the holodeck as introduced by *Star Trek* [STAR 1982] or a super form of a *CAVE* (Cave Automatic Virtual Environment) as it is also named. That is, a computer generated completely virtual part of the world, that is more or less indistinguishable from reality. Or, as an intermediate step, computer generated images, that are truly *photo-realistic*.

Besides the huge steps made in computer hardware and algorithm development since the beginning of computers, there is no holodeck yet. Therefore, also the topic of cloth visualization research is not finished.

In the following, I will give pointers to future research directions for the various aspects I presented in the last chapters.

Perception

I want to start with the general topic of *perception*. That is, besides all the effort put into generating highly realistic images, what is actually perceived by the human observer? Understanding the physiological interactions between the human visual system and the brain and understanding the kind of filters which are applied to all the information overflow which the human observer is bombarded with, might lead to important insights, which parts of an image are *really* important. Perhaps, as the introduced experiment in this thesis suggests, a lot of computational effort is not necessary to convince the average human observer, that the image is photo-realistic or correct.

Therefore, I believe besides thinking about more advanced and efficient algorithms it is of equal importance to perform perception experiments, for example also in the context of material reflection properties, to achieve a much greater performance increase.

Shadows

Self-shadowing, as stated above, is a computational complex problem, due to the fact that each surface point has to be evaluated. I think that further research should focus on parallel processing and the combination of the calculation with the overall perception of shadows. Parallelism could be achieved using the advancing capabilities of modern graphics hardware.

Animation

As in this work described, the transfer of compression schemes from one topic (material surface properties) to another (animation sequences) might be successful. Therefore, computer graphics should furthermore try to adopt existing compression algorithms to fields of applications, where they were not used before. The introduced clustered PCA compression might also be combined with other schemes like linear prediction coding or wavelet compression.

Material Reflection Properties

I think this work and all related work in the group made clear, that using measured real-world material reflection properties is superior to any other form of artificial generated data, when it comes to create a realistic *look & feel*. The drawback of the huge data amount and the coupled compression of this data was already tackled within this work and of course is already topic of current research.
It might be useful to investigate other signal processing algorithms and apply them to the data to ultimately decouple not only geometry and reflection but also mesostructure effects and the *core* material reflection properties.

Cloth Visualization

While the above mentioned topics are more general in nature, some future work directions in the field of cloth visualization itself arose for example from the *Virtual Try-On* project (see Chapter 6).
To further enhance the visual acceptance of the rendered cloth, more subtle details should be added. That is, knobs, zippers or pockets should also be measured and integrated into the rendering pipeline. This also includes the further measurement and integration of the silhouette enhancements (see 5.7).

ANHANG A Miscellaneous

This appendix includes a list of all videos, figures and tables throughout this thesis. In addition, the used data sources are referenced.

This is followed by the list of all references, a short curriculum vitae and the publication list.

Videos

This chapter lists all videos with a short description accompanying this thesis. The following icon marks parts where additional video material is available.

I	BTF acquisition setup	[2:39 min]	2.1.9
II	Animation sequence generation	[6:13 min]	3.2.3
III	Animation compression: sequence	[0:14 min]	3.3.3
IV	Animation compression	[1:53 min]	3.3.3
V	Self-shadowing: static results	[3:17 min]	4.3.7
VI	Self-shadowing: dynamic results	[9:25 min]	4.4.7
VII	Materials: textures vs. BTF	[0:54 min]	5.2
VIII	Materials: rendering results	[4:31 min]	5.6
IX	VTO results	[2:23 min]	6.2.1

Data Sources

This chapter includes the used external data sources in this work and their references.

The textiles shown in this work were generated with the cloth simulation engine TüTex and provided by M. Wacker, M. Keckeisen, and S. Kimmerle from the WSI/GRIS at the University of Tübingen. For further details refer to [Etzmuss *et al.* 2003; Kimmerle *et al.* 2003; Mezger *et al.* 2003] and the *Virtual Try-On* project [VTO 2005].

The Stanford *bunny* model is courtesy of the 3D scanning repository, University of Stanford [Turk & Levoy 1994].

The stanford *dragon* model is courtesy of the 3D scanning repository, University of Stanford.

The *chicken* sequence is property of Microsoft Inc. and was kindly provided by John Snyder.

The *Igea artifact* model is courtesy of Cyberware [Cyberware 2005].

The *avatar* sequence was created by the University of Tuebingen in the focus of the Virtual Try-On project [VTO 2005].

$POSER$® from e-frontier, Inc. was used for the creation of the *skeleton* sequence [e-frontier 2005].

The *cow* animation was created by Matthias Müller (ETH Zürich).

The *dolphin* sequence was kindly provided by Zachi Karni.

The *face* sequence was kindly provided by Zachi Karni.

$POSER$® from e-frontier, Inc. and demonstrational motion capture data were used for the creation of the *dance* sequence [e-frontier 2005].

$POSER$® from e-frontier, Inc. and inbuild facial expressions were used for the creation of the *head* sequence [e-frontier 2005].

List of Figures

List of Tables

References

ABYSS. (1989). *The Abyss.*
Fox Studios. 93

AGARWAL, S., RAMAMOORTHI, R., BELONGIE, S. AND JENSEN, H. (2003). Structured Importance Sampling of Environment Maps. *Transactions on Graphics*, **22**, 605–612. 182

AGRAWALA, M., RAMAMOORTHI, R., HEIRICH, A. AND MOLL, L. (2000). Efficient image-based methods for rendering soft shadows. *Pages 375–384 of: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. 84

AILA, T. AND AKENINE-MÖLLER, T. (2004). A Hierarchical Shadow Volume Algorithm. *Pages 15–23 of: Proceedings of Graphics Hardware 2004.* Eurographics Association. 78

AILA, T. AND LAINE, S. (2004). Alias-Free Shadow Maps. *Pages 161–166 of: Proceedings of Eurographics Symposium on Rendering 2004.* Eurographics Association. 82

AKENINE-MÖLLER, T. AND ASSARSSON, U. (2002). Approximate soft shadows on arbitrary surfaces using penumbra wedges. *Pages 297–306 of: Proceedings of the 13th Eurographics workshop on Rendering.* Eurographics Association. 84, 156

AKENINE-MÖLLER, T. AND HAINES, E. (2002). *Real-Time Rendering, 2nd edition.* A.K. Peters Ltd. 59, 61, 78, 83, 207

ALEXA, M. AND MÜLLER, W. (2000). Representing Animations by Principal Components. *Computer Graphics Forum*, **19**(3), 411–418. 120, 137

ALLIEZ, P. AND DESBRUN, M. (2001). Progressive compression for lossless transmission of triangle meshes. *Pages 195–202 of: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press. 137

ASHIKHMIN, M., PREMOZE, S. AND SHIRLEY, P. (2000). A Microfacet-based BRDF Generator. *In: ACM Siggraph 2000 Conference Proceedings*. 196

ASSARSSON, U. AND AKENINE-MÖLLER, T. (2003). A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Trans. Graph.*, **22**(3), 511–520. 84, 156

ASSARSSON, U., DOUGHERTY, M., MOUNIER, M. AND AKENINE-MÖLLER, T. (2003). An optimized soft shadow volume algorithm with real-time performance. *Pages 33–40 of: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. Eurographics Association. 156, 157

BARAFF, D. AND WITKIN, A. (1998). Large steps in cloth simulation. *Pages 43–54 of: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press. 105, 106

BARAFF, D., WITKIN, A. AND KASS, M. (2003). Untangling cloth. *ACM Trans. Graph.*, **22**(3), 862–870. 107

BERGMANN, L. AND SCHAEFER, C. (2004). *Lehrbuch der Experimentalphysik, Band 3, Optik, Wellen- und Teilchenoptik*. 10. edn. de Gruyter. 9

BERTHOLD, M. AND HAND, D. J. (eds). (2003). *Intelligent Data Anakysis, An Introduction, 2nd edition*. Springer-Verlag. 109, 112

BILODEAU, B. AND SONGY, M. (1999). Real Time Shadows. *In: Creativity '99*. 77, 207

BIPM 2006. *http://www.bipm.fr/en/si/*.
Bureau International des Poids et Mesures. 9

BITTNER, J., WIMMER, M., PIRINGER, H. AND PURGATHOFER, W. (2004). Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Pages 615–624 of: Proceedings of Eurographics*. 74

BLINN, J. F. AND NEWELL, M. E. (1976). Texture and reflection in computer generated images. *Communications of the ACM*, **19**, 542–547. 60, 178, 197

BLINN, J. F. (1977). Models of Light Reflection for Computer Synthesized Pictures. *Computer Graphics*, **11**(3), 192–198. 31

BLINN, J. F. (1978). Simulation of Wrinkled Surfaces. *Pages 286–292 of: Computer Graphics (SIGGRAPH '78 Proceedings)*, vol. 12. 20, 220

BORODIN, P., GUMHOLD, S., GUTHE, M. AND KLEIN, R. (2003)(September). High-Quality Simplification with Generalized Pair Contractions. *Pages 147–154 of: Proceedings of GraphiCon'2003*. 62, 157

BORSHUKOV, G. (2003). Measured BRDF in Film Production - Realistic Cloth Appearance for 'The Matrix Reloaded'. *In: SIGGRAPH 2003 Sketches*. 99

BOWDEN, R. (2000). Learning statistical models of human motion. *In: ICVPR2000, IEEE Workshop on Human Modelling, Analysis and Synthesis*. 120

BRABEC, S. AND SEIDEL, H. P. (2001). Hardware-accelerated Rendering of Antialiased Shadows with Shadow Maps. *Pages 209–214 of: Proceedings of the Computer Graphics International 2001*. 83

BRABEC, S. AND SEIDEL, H.-P. (2002). Single sample soft shadows using depth maps. *In: Proceedings of the Graphics Interface (GI) 2002*. 84

BRABEC, S. AND SEIDEL, H.-P. (2003). Shadow Volumes on Programmable Graphics Hardware. *In: Proceedings EUROGRAPHICS 2003*. 66, 78, 207, 221

BREEN, D. E., HOUSE, D. H. AND WOZNY, M. J. (1994). Predicting the drape of woven cloth using interacting particles. *Pages 365–372 of: SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press. 103

BRENNAN, C. (2002). *ShaderX: Vertex and Pixel Shaders Tips and Tricks*. Wordware. Chap. Shadow Volume Extrusion using a Vertex Shader, pages 188–192. 207, 221

BRICENO, H. M., SANDER, P. V., MCMILLAN, L., GORTLER, S. AND HOPPE, H. (2003). Geometry videos: a new representation for 3D animations. *Pages 136–146 of: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association. 137

BRUDERLIN, A. AND CALVERT, T. W. (1989). Goal-directed, dynamic animation ofhuman walking. *Pages 233–242 of: Proceedings of SIGGRAPH 89*, vol. 23. 120

BRUDERLIN, A. AND CALVERT, T. W. (1996). Knowledge-driven, interactive animation of human running. *Pages 213–221 of: Graphics Interface '96*. 120

BTFDBB 2005. *http://btf.cs.uni-bonn.de*.
BTF Database Bonn. 23

BUNNELL, M. (2005). Dynamic Ambient Occlusion and Indirect Lighting. *In: GPU Gems 2*. Addison-Wesley. 90

BVH 2006. *http://www.wotsit.org*.
Wotsit Data Formats. 98

CAM 2006. *http://www.cocs.com/poser/movies.htm*.
History of Computer Animation in the Movies. 93

CARIGNAN, M., YANG, Y., THALMANN, N. M. AND THALMANN, D. (1992). Dressing animated synthetic actors with complex deformable clothes. *Pages 99–104 of: SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press. 100

CG 2006. *http://developer.nvidia.com/page/cg_main.html*.
NVidia CG. 73

CHAN, E. AND DURAND, F. (2003). Rendering Fake Soft Shadows with Smoothies. *Pages 208–218 of: Proceedings of the Eurographics Symposium on Rendering*. Eurographics Association. 84, 156

CHEN, W.-C., BOUGUET, J.-Y., CHU, M. H. AND GRZESZCZUK, R. (2002). Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. *In: Proceedings of ACM SIGGRAPH 2002*. 46, 196, 202, 207

CHI, D. M., COSTA, M., ZHAO, L. AND BADLER, N. (2000). The emote model for effort and shape. *Pages 173–182 of: Siggraph 2000*. 120

CHOI, K.-J. AND KO, H.-S. (2002). Stable but responsive cloth. *Pages 604–611 of: SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press. 105

CIE. (2004). *Colorimetry*. Tech. rept. Commission Internationale de l'Éclairage. 11

CIE 2006. *http://www.cie.co.at/cie/*.
CIE, Commission Internationale de l'Éclairage. 11

CIE1987. *International Lighting Vocabulary, Publication Commission Internationale de l'Eclairage (CIE) 17.4*. ISBN 3 900 734 0 70. 9, 12

COHEN, M. F. AND GREENBERG, D. P. (1985). The hemi-cube: a radiosity solution for complex environments. *Pages 31–40 of: SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. ACM Press. 51, 90, 164, 170, 178, 210

COHEN, M. F. AND WALLACE, J. R. (1993). *Radiosity and Realistic Image Syynthesis*. Morgan Kaufmann Publishers, Inc. 164

COLLIER, J.R., COLLIER, B.J., O'TOOLE, G. AND SARGAND, S.M. (1991). Drape Prediction by Means of Finite-Element Analysis. *Journal of the Textile Institute*, **82**, 1. 102

COOK, R. L. (1984). Shade trees. *Pages 223–231 of: SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press. 21

COOK, R. L., PORTER, T. AND CARPENTER, L. (1984). Distributed ray tracing. *Pages 137–145 of: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. ACM Press. 42

COOMBE, G., HARRIS, M. J. AND LASTRA, A. (2004). Radiosity on Graphics Hardware. *In: Graphics Interface*. 91, 178

CROW, F. C. (1977). Shadow algorithms for computer graphics. *Pages 242–248 of: SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*. ACM Press. 76, 156, 178, 207, 221

CURET 2005. *http://www1.cs.columbia.edu/CAVE/curet/*.
Columbia-Utrecht Reflectance and Texture Database. 23, 198

CURLESS, B. AND LEVOY, M. (1996). A Volumetric Method for Building Complex Models from Range Images. *Computer Graphics*, **30**(Annual Conference Series), 303–312. 157

CYBERWARE 2005. *http://www.cyberware.com/*.
Cyberware. 248

DACHSBACHER, C. AND STAMMINGER, M. (2003). Translucent Shadow Maps. *Pages 197–201 of: Rendering Techniques*. 82

DANA, K. J., NAYAR, S. K., VAN GINNEKEN, B. AND KOENDERINK, J. J. (1999a). 3D Textured Surface Modeling. *In: WIAGMOR Workshop CVPR '99*. 199

DANA, K. J., VAN GINNEKEN, B., NAYAR, S. K. AND KOENDERINK, J. J. (1999b). Reflectance and Texture of Real-World Surfaces. *ACM Transactions on Graphics*, **18**, 1–34. 16, 22, 194, 197, 221

DANA, K. J. AND WANG, J. (2004). Device for convenient measurement of spatially varying bidirectional reflectance. *Journal of the Optical Society of America*, **A**(January), 1–12. 26

DAUBERT, K. AND SEIDEL, H.-P. (2002). Hardware-Based Volumetric Knit-Wear. *Computer Graphics Forum 21(3) - Proceedings of Eurographics*, **63**(2), 314–325. 21, 108, 193, 195, 221

DAUBERT, K., LENSCH, H. P. A., HEIDRICH, W. AND SEIDEL, H.-P. (2001). Efficient Cloth Modeling and Rendering. *Pages 63–70 of: 12th Eurographics Workshop on Rendering*. 32, 45, 50, 54, 108, 195, 221

DEBEVEC, P., HAWKINS, T., TCHOU, C., DUIKER, H.-P., SAROKIN, W. AND SAGAR, M. (2000). Acquiring the Reflectance Field of a Human Face. *Pages 145–156 of: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 18, 196

DEBEVEC, P. (1998). Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics With Global Illumination and High Dynamic Range Photography. *Pages 189–198 of:* COHEN, M. (ed), *SIGGRAPH 98 Conference Proceedings*. Annual Conference SeriesAddison Wesley, for ACM SIGGRAPH. ISBN 0-89791-999-8. 58, 185, 197, 209, 217

DEBEVEC, P., WARD, G. AND LEMMON, D. (2003)(June). HDRI and Image-Based Lighting. *In: Proceedings of SIGGRAPH 2003: Course Notes 19*. 58

DEBEVEC, P., REINHARD, E., WARD, G. AND PATTANAIK, S. (2004)(June). High Dynamic Range Imaging. *In: Proceedings of SIGGRAPH 2004: Course Notes 13*. 58

DEBEVEC, P. E. AND MALIK, J. (1997). Recovering High Dynamic Range Radiance Maps from Photographs. *Pages 369–378 of:* WHITTED, T. (ed), *SIGGRAPH 97 Conference Proceedings.* Annual Conference SeriesAddison Wesley, for ACM SIGGRAPH. 48, 58, 185, 209

DEERING, M. (1995). Geometry compression. *Pages 13–20 of: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques.* ACM Press. 137

DELOURA, M. A. (ed). (2001). *Game Programming Gems 2.* Charles River Media Inc. 207

DESBRUN, M., SCHRÖDER, P. AND BARR, A. (1999). Interactive Animation of Structured Deformable Objects. *Pages 1–8 of: Graphics Interface.* 101, 105

DILLON, W. AND GOLDSTEIN, M. (1984). *Multivariate Analysis: Methods and Applications.* Wiley and Sons. 109, 110

DISCHLER, J. M. (1998). Efficiently Rendering Macro Geometric Surface Structures with Bi-Directional Texture Functions. *In: Proceedings of the Eurographics Workshop on Rendering.* 16

E-FRONTIER 2005. *http://www.e-frontier.com/.*
e-frontier, Inc. 100, 249

EBERHARDT, B., WEBER, A. AND STRASSER, W. (1996). A Fast, Flexible, Particle-System Model for Cloth Draping. *IEEE Comput. Graph. Appl.*, **16**(5), 52–59. 103

EBERLE, D., HADAP, S., ERICSON, C., LIN, M. C., REDON, S. AND VOLINO, P. (2004). Collision Detection and Proximity Queries. *In: Siggraph 2004 Course Notes 14.* ACM Siggraph. 107

EISCHEN, J. W., DENG, S. AND CLAPP, T. G. (1996). Finite-Element Modeling and Control of Flexible Fabric Parts. *IEEE Comput. Graph. Appl.*, **16**(5), 71–80. 102

ETZMUSS, O., KECKEISEN, M. AND STRASSER, W. (2003). A fast finite element solution for cloth modelling. *In: PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications.* 247

EVERITT, C. AND KILGARD, M. J. (2002). Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. *In: NVIDIA White paper.* 78, 207

FALOUTSOS, P., VAN DE PANNE, M. AND TERZOPOULOS, D. (2001a). Composable Controllers for Physics-Based Character Animation. *Pages 251–260 of: SIGGRAPH 2001 Proceedings*. 120

FALOUTSOS, P., VAN DE PANNE, M. AND TERZOPOULOS, D. (2001b). The virtual stuntman: dynamic characters with a repertoire of autonomous motor skills. *Computers & Graphics*, **6**(25), 933–953. 120

FEJES TOTH, G. (1972). *Lagerungen in der Ebene, auf der Kugel und im Raum, 2nd ed.* Springer-Verlag. 183

FERNANDO, R., FERNANDEZ, S., BALA, K. AND GREENBERG, D. P. (2001). Adaptive shadow maps. *Pages 387–390 of: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press. 82

FILIP, J. AND HAINDL, M. (2004). Non-linear Reflectance Model for Bidirectional Texture Function Synthesis. *In: ICPR 2004*. 33, 45, 54

FOLEY, J. D., VAN DAM, A., FEINER, S. K. AND HUGHES, J. F. (1996). *Fundamentals of Interactive Computer Graphics. 2nd Edition in C*. Addison-Wesley. 13, 92

FURUKAWA, R., KAWASAKI, H., IKEUCHI, K. AND SAKAUCHI, M. (2002). Appearance based object modeling using texture database: acquisition, compression and rendering. *Pages 257–266 of: Proceedings of the 13th Eurographics workshop on Rendering*. Eurographics Association. 22

GAN, L., LY, N. G. AND STEVEN, G. P. (1991). A Finite Element Analysis of the Draping of Fabrics. *In: Proc. 6th Int'l Conf. in Australia on Finite Element Methods*. 102

GLEICHER, M. (2001). Comparing constraint-based motion editing methods. *Graphical Models*, **2**(63), 107–123. 120

GLSL 2006. *http://www.opengl.org/documentation/oglsl.html*.
GL Shading Language. 73

GOESELE, M., LENSCH, H. P., LANG, J., FUCHS, C. AND SEIDEL, H.-P. (2004)(August). DISCO-Acquisition of Translucent Objects. *In: Proceedings of SIGGRAPH 2004*. 16

GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P. AND RIESENFELD, R. (1999). Interactive technical illustration. *Pages 31–38 of: SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics.* New York, NY, USA: ACM Press. 85

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R. AND COHEN, M. F. (1996). The Lumigraph. *Computer Graphics*, **30**(Annual Conference Series), 43–54. 17, 18, 196

GREENE, N. (1986). Environment Mapping and Other Applications of World Projection. *IEEE Computer Graphics and Applications*, **6**(11), 21–29. 48, 61, 185, 197

GRÖLLER, E., RAU, R. AND STRASSER, W. (1995). Modeling and Visualization of Knitwear. *IEEE Transactions on Visualization and Computer Graphics*, **1**(4), 302–310. 21, 108, 193, 195, 221

GRÖLLER, E., RAU, R. AND STRASSER, W. (1996). Modeling textiles as three dimensional textures. *Pages 205–ff. of: Proceedings of the eurographics workshop on Rendering techniques '96.* Springer-Verlag. 108, 195, 221

GRZESZCZUK, R., TERZOPOULOS, D. AND HINTON, G. (1998). NeuroAnimator: fast neural network emulation and control of physics-based models. *Pages 9–20 of: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM Press. 105

GU, X., GORTLER, S. J. AND HOPPE, H. (2002). Geometry images. *Pages 355–361 of: Proceedings of the 29th annual conference on Computer graphics and interactive techniques.* ACM Press. 137

GUMHOLD, S. AND STRASSER, W. (1998). Real Time Compression of Triangle Mesh Connectivity. *Pages 133–140 of: SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques.* 137

GUSKOV, I. AND KHODAKOVSKY, A. (2004). Wavelet Compression of Parametrically Coherent Mesh Sequences. *In: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 137, 149

GUTHE, M., BORODIN, P. AND KLEIN, R. (2005). Fast and accurate Hausdorff distance calculation between meshes. *Journal of WSCG*, **13**(2), 41–48. 62

HAINES, E. (2001). Soft planar shadows using plateaus. **6**(1), 19–27. 84

HALL, J. D. AND HART, J. C. (2004). *GPU Acceleration of Iterative Clustering*. Manuscript accompanying poster at GP$\hat{2}$: The ACM Workshop on General Purpose Computing on Graphics Processors, and SIGGRAPH 2004 poster. 138

HAN, J. Y. AND PERLIN, K. (2003). Measuring bidirectional texture reflectance with a kaleidoscope. *ACM Trans. Graph.*, **22**(3), 741–748. 27

HART, D., DUTRE, P. AND GREENBERG, D. (1999). Direct illumination with lazy visibility evaluation. *In: SIGGRAPH 99 Conference Proceedings*. 83, 90, 179

HART, E., GOSSELIN, D. AND ISIDORO, J. (2001). Vertex Shading with Direct3D and OpenGL. *In: Game Developers Conference 2001*. 207, 221

HASENFRATZ, J., LAPIERRE, M., HOLZSCHUCH, N. AND SILLION, F. (2003). A survey of Real-Time Soft Shadows Algorithms. *In: Eurographics*Eurographics, for Eurographics. State-of-the-Art Report. 43, 83, 156, 178

HAUTH, M., ETZMUSS, O., EBERHARDT, B., KLEIN, R., SARLETTE, R., SATTLER, M., DAUBERT, K. AND KAUTZ, J. (2002). Cloth Animation and Rendering. *In: Eurographics 2002 Tutorial Notes T3*. The Eurographics Association. 5, 23, 38, 100

HDRSHOP 2005. *http://www.debevec.org/HDRShop/*.
HDRShop. 58, 61, 185

HECKBERT, P. S. AND HERF, M. (1997)(Jan.). *Simulating Soft Shadows with Graphics Hardware*. Tech. rept. CMU-CS-97-104. CS Dept, Carnegie Mellon U. 83

HEIDMANN, T. (1991). Real shadows, real time. *Iris Universe*, **18**, 28–31. 77, 207

HEIDRICH, W. AND SEIDEL, H.-P. (1998). View-Independent Environment Maps. *In: Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '98*. 46, 61

HEIDRICH, W. AND SEIDEL, H.-P. (1999). Realistic, Hardware-Accelerated Shading and Lighting. *In:* ROCKWOOD, A. (ed), *Siggraph 1999, Annual Conference Proceedings*. Annual Conference Series. Addison Wesley Longman. 185, 220

Heidrich, W., Brabec, S. and Seidel, H.-P. (2000). Soft Shadow Maps for Linear Lights. *Pages 269–280 of: Proceedings of the Eurographics Workshop on Rendering Techniques 2000.* London, UK: Springer-Verlag. 84

Hind, K. and McCartney, J. (1990). Interactive Garment Design. *Visual Computer*, **6**, 53–61. 102

Hoppe, H. (1996). Progressive meshes. *Pages 99–108 of: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.* ACM Press. 137

Hourcade, J. C. and Nicolas, A. (1985). Algorithms for Antialiased Cast Shadows. *Computers and Graphics*, **9**(3), 259–265. 82

House, D. and Breen, D. (2000). *Cloth Modeling and Animation.* A K Peters. 54, 92, 100

Hu, H. H., Gooch, A. A., Thompsona, W. B., Smits, B. E., Shirley, P. and Rieser, J. J. (2000). Visual Cues for Imminent Object Contact in Realistic Virtual Environments. *In: IEEE Visualization 2000.* 156

Hyeong-Seok, K., David, B., Michael, H., Ronald, F. and Rob, H. (2003). Clothing Simulation and Animation. *In: Siggraph 2003 Course Notes 29.* ACM Siggraph. 101

Hyeong-Seok, K., Kwang-Jin, C., Fedkiw, R. and Dongliang, Z. (2005). Advanced Topics on Advanced Topics on Clothing Simulation and Clothing Simulation and Animation. *In: Siggraph 2003 Course Notes 6.* ACM Siggraph. 101

Ibarria, L. and Rossignac, J. (2003). Dynapack: space-time compression of the 3D animations of triangle meshes with fixed connectivity. *Pages 126–135 of: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* Eurographics Association. 137

ID 2006. *http://www.idsoftware.com/.*
Doom3, id software. 78

Iones, A., Krupkin, A., Sbert, M. and Zhukov, S. (2003). Fast realistic lighting for video games. *IEEE Computer Graphics & Applications*, **23**, 54–64. 89, 179

Isidoro, J. and Mitchell, J. L. (2002). User-Customizable Real-Time Fur. *In: SIGGRAPH 2002-Sketch.* 220

JAMES, D. L. AND FATAHALIAN, K. (2003). Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph.*, **22**(3), 879–887. 105

JENSEN, H. W. (2001). *Realistic Image Synthesis Using Photon Mapping*. AK Peters. 15, 75, 162, 169, 210

JENSEN, H.W. (1996). Global Illumination using Photon Maps. *In: Eurographics Workshop on Rendering 1996*. 42

JOLLIFFE, I. (1986). *Principal Component Analysis*. Springer-Verlag. 109, 111, 121, 203

JUKKA ARVO, MIKA HIRVIKORPI AND JOONAS TYYSTJÄRVI. (2004). Approximate Soft Shadows Using Image-Space Flood-Fill Algorithm. *Computer Graphics Forum*, **23**(3), 271–280. 84

KAJIYA, J. T. (1986). The rendering equation. *Pages 143–150 of: SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. ACM Press. 42, 162, 169, 178

KAJIYA, J. T. AND KAY, T. L. (1989). Rendering Fur with Three Dimensional Textures. *In: SIGGRAPH 89 Proceedings*. 220, 221

KAMBHATLA, N. AND LEEN, T.K. (1997). Dimension Reduction by Local PCA. *Neural Computation*, **9**, 1493–1516. 38, 53, 115, 135, 140

KARNI, Z. AND GOTSMAN, C. (2000). Spectral Compression of Mesh Geometry. *Pages 279–286 of:* AKELEY, K. (ed), *Siggraph 2000, Computer Graphics Proceedings*. ACM Press / ACM SIGGRAPH / Addison Wesley Longman. 137, 138

KARNI, Z. AND GOTSMAN, C. (2004). Compression of Soft-Body animation sequences. *Computer and Graphics*, **28**, 25–34. 137, 142, 143, 149

KARNI, Z., BOGOMJAKOV, A. AND GOTSMAN, C. (2002). Efficient compression and rendering of multi-resolution meshes. *Pages 347–354 of: Proceedings of the conference on Visualization '02*. IEEE Computer Society. 137

KATZ, S. AND TAL, A. (2003). Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph.*, **22**(3), 954–961. 138, 141

KAUTZ, J. AND MCCOOL, M. (1999). Interactive Rendering with Arbitrary BRDFs using Separable Approximations. *In: 10th Eurographics Workshop on Rendering*. 35, 45, 196

KAUTZ., J., VÁZQUEZ, P.-P., HEIDRICH, W. AND SEIDEL, H.-P. (2000). A Unified Approach to Prefiltered Environment Maps. *In: Proc. 11th Eurographics Workshop on Rendering*. 185

KAUTZ, J., LEHTINEN, J. AND AILA, T. (2004). Hemispherical Rasterization for Self-Shadowing of Dynamic Objects. *Pages 179–184 of: Proceedings of Eurographics Symposium on Rendering 2004*. 90, 179

KAUTZ, J. AND MCCOOL, M. D. (2000). Approximation of Glossy Reflection with Prefiltered Environment Maps. *Pages 119–126 of: Graphics Interface 2000*. 48, 49, 197

KAUTZ, J., LEHTINEN, J. AND SLOAN, P.-P. (2005). Pre-Computed Radiance Transfer: Theory and Practice. *In: Proceedings of ACM SIGGRAPH 2005: Course Notes*. 43, 91

KAWABATA, S. (1980). *The Standardization and Analysis of Hand Evaluation, 2nd Edition*. The Textile Machinery Society of Japan. 102

KELLER, A. (1997). Instant Radiosity. *In: Computer Graphics Proceedings SIGGRAPH 97*. 179

KENDALL, M. (1975). *Multivariate Analysis*. Charles Griffin Co. 111, 121, 203

KERSTEN, D., KNILL, D., MAMASSIAN, P. AND BÜLTHOFF, I. (1996). Illusory motion from shadows. *Nature*, **379**, 31. 156

KERSTEN, D., MAMASSIAN, P. AND KNILL, D.C. (1997). Moving cast shadows induce apparent motion in depth. *Pages 171–192 of: Perception*, vol. 26. 156

KHODAKOVSKY, A., SCHRÖDER, P. AND SWELDENS, W. (2000). Progressive geometry compression. *Pages 271–278 of: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 137

KILGARD, M. J. (2001). More Advanced Hardware Rendering Techniques. *In: Game Developers Conference 2001*. 207

KILGARD, M. J. (2002). Shadow Mapping with Todays OpenGL Hardware. *In: SIGGRAPH 2002 Course*. 90, 156, 179, 187, 207

KIMMERLE, S., KECKEISEN, M., MEZGER, J. AND WACKER, M. (2003). Tütex: A cloth modelling system for virtual humans. *In: Proceedings 3D Modelling*. 247

KIRSANOV, D., SANDER, P. V. AND GORTLER, S. J. (2003). Simple silhouettes for complex surfaces. *Pages 102–106 of: SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing.* Eurographics Association. 66

KLEIN, R., LIEBICH, G. AND STRASSER, W. (1996). Mesh Reduction with Error Control. *In:* YAGEL, R. AND NIELSON, G. M. (eds), *IEEE Visualization 96.* ACM Press. 63

KNUTH, M. AND FUHRMANN, A. (2005). Self-Shadowing of dynamic scenes with environment maps using the GPU. *Pages 31–38 of: WSCG FULL papers proceedings.* 91

KOENDERINK, J. J. (1984). What does the occluding contour tell us about solid shape. *Perception*, **3**(13), 321–330. 220

KOLLIG, T. AND KELLER, A. (2003). Efficient Illumination by High Dynamic Range Images. *In: Proceedings Eurographics Symposium on Rendering 2003.* 182

KONTKANEN, J. AND LAINE, S. (2005). Ambient Occlusion Fields. *Pages 41–48 of: Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games.* ACM Press. 91

KOUDELKA, M. L., MAGDA, S., BELHUMEUR, P. N. AND KRIEGMAN, D. J. (2003). Acquisition, Compression and Synthesis of Bidirectional Texture Functions. *In: 3rd International Workshop on Texture Analysis and Synthesis.* 26, 36

LAFORTUNE, E. P. F., FOO, S.-C., TORRANCE, K. E. AND GREENBERG, D. P. (1997). Non-Linear Approximation of Reflectance Functions. *Pages 117–126 of: Proc. SIGGRAPH 97.* 31, 196

LANDIS, H. (2002). Production-Ready Global Illumination. *In: 'RenderMan in Production' SIGGRAPH Course Notes 2002.* 90, 179

LAPACK 2005. *http://www.netlib.org/lapack.*
LAPACK. 109, 141

LARBOULETTE, C. AND CANI, M.-P. (2004). Real-Time Dynamic Wrinkles. *In: Computer Graphics International.* IEEE Computer Society Press. Greece. 105

LARSON, G. W. AND SHAKESPEARE, R. A. (1998). *Rendering with Radiance The Art and Science of Lighting Visualization*. Morgan Kaufmann Publishers Inc. 57

LARSON, G. J. W. (1992). Measuring and Modeling Anisotropic Reflection. *In: Proc. SIGGRAPH 92*. 17, 31, 196

LASSETER, J. (1987). Principles of Traditional Animation Applied to 3D Computer Animation. *Computer Graphics (SIGGRAPH 87)*, **4**(21), 35–44. 93

LASZLO, J. F., VAN DE PENNE, M. AND FIUME, E. L. (1996). Limit cycle control and its application to the animation of balancing and walking. *In: Proceedings of Siggraph 96*. 120

LATTA, L. AND KOLB, A. (2002). Homomorphic factorization of BRDF-based lighting computation. *Pages 509–516 of: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press. 197

LEE, J., CHAI, J., REITSMA, P., HODGINS, J. AND POLLARD, N. (2002). Interactive control of avatars animated with human motion data. *In: Proc. SIGGRAPH 2002*. 119, 120, 122, 128

LENGYEL, J., PRAUN, E., FINKELSTEIN, A. AND HOPPE, H. (2001). Real-time fur over arbitrary surfaces. *Pages 227–232 of: Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM Press. 220, 224

LENGYEL, J. E. (1999). Compression of time-dependent geometry. *Pages 89–95 of: Proceedings of the 1999 symposium on Interactive 3D graphics*. ACM Press. 137, 140

LENSCH, H. P. A., GOESELE, M., CHUANG, Y.-Y., ITIM HAWKINS, MARSCHNER, S., MATUSIK, W. AND MUELLER, G. (2005). Realistic Materials in Computer Graphics. *In: Siggraph 2005 Course Notes 10*. ACM Siggraph. 108

LENSCH, H. P., KAUTZ, J., GOESELE, M., HEIDRICH, W. AND SEIDEL, H.-P. (2001). Image-Based Reconstruction of Spatially Varying Materials. *In: Proceedings of Eurographics Rendering Workshop ´01*. 22, 196

LEVOY, M. AND HANRAHAN, P. (1996). Light Field Rendering. *Computer Graphics*, **30**(Annual Conference Series), 31–42. 17, 18, 196

LIGHTPROBES 2005. *http://www.debevec.org/Probes/*.
Light Probe Image Gallery. 48, 185

LIN, M. C. AND CANNY, J. F. (1992). Efficient collision detection for animation. *In: In Eurographics Workshop on Simulation and Animation*. 106

LIU, X., YU, Y. AND SHUM, H.-Y. (2001). Synthesizing bidirectional texture functions for real-world surfaces. *Pages 97–106 of: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 198

LIU, X., HU, Y., ZHANG, J., TONG, X., GUO, B. AND SHUM, H.-Y. (2004). Synthesis and Rendering of Bidirectional Texture Functions on Arbitrary Surfaces. *IEEE Transactions on Visualization and Computer Graphics*, **10**(3), 278–289. 36, 47, 54

LLOYD, S. P. (1982). Least square quantization in PCM. *IEEE Transactions on Information Theory*, **28**(2), 129–137. 136

LOKOVIC, T. AND VEACH, E. (2000). Deep shadow maps. *Pages 385–392 of: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 82

LOOP, C. T. (1987). *Smooth subdivision surfaces based on triangles*. M.Phil. thesis, Univ. of Utah, Dept. of Mathematics. 63

LOTR. (2003). *Lord of the rings trilogy*.
New Line Cinema. 93, 119

LUEBKE, D. (2001). A Developers Survey of Polygonal Simplification Algorithms. *Pages 24–35 of: Computer Graphics & Applications*, vol. 23. 63

MACQUEEN, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Pages 281–297 of: Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press. 114

MADISON, C., THOMPSON, W. B., KERSTEN, D. J., SHIRLEY, P. AND SMITS, B. S. (2001). Use of Interreflection and Shadow for Surface Contact. *In: Perception and Psychophysics*, vol. 63. 156

MAGNENAT-THALMANN, N., CORDIER, F., KECKEISEN, M., KIMMERLE, S., KLEIN, R. AND MESETH, J. (2004). Simulation of Clothes for Real-time Applications. *In:* MAGNENAT-THALMANN, N., VOLINO, P., THOMASZEWSKI, B. AND WACKER, M. (eds), *Eurographics 2004 Tutorial Notes T1*. The Eurographics Association. 100

MAGNENAT-THALMANN, N., VOLINO, P., THOMASZEWSKI, B. AND WACKER, M. (2005). Key techniques for interactive virtual garment simulation. *In:* MAGNENAT-THALMANN, N., VOLINO, P., THOMASZEWSKI, B. AND WACKER, M. (eds), *Eurographics 2005 Tutorial Notes T4*. The Eurographics Association. 100

MALZBENDER, T., GELB, D. AND WOLTERS, H. (2001). Polynomial texture maps. *Pages 519–528 of: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 19, 27, 33, 196

MARKOSIAN, L., KOWALSKI, M. A., GOLDSTEIN, D., TRYCHIN, S. J., HUGHES, J. F. AND BOURDEV, L. D. (1997). Real-time nonphotorealistic rendering. *Pages 415–420 of: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 66, 156, 221

MARSCHNER, S. R., WESTIN, S. H., LAFORTUNE, E. P. F., TORRANCE, K. E. AND GREENBERG, D. P. (1999). Image-based BRDF Measurement Including Human Skin. *Pages 139–152 of: Proceedings of EGRW '99*. 17

MASSELUS, V., PEERS, P., DUTRE;, P. AND WILLEMS, Y. D. (2003). Relighting with 4D incident light fields. *ACM Trans. Graph.*, **22**(3), 613–620. 19

MATRIX. (2003). *The Matrix trilogy*.
Warner Bros. 99, 119

MATROX 2006. *http://www.matrox.com*.
Matrox. 21

MATUSIK, W., PFISTER, H., NGAN, A., BEARDSLEY, P., ZIEGLER, R. AND MCMILLAN, L. (2002). Image-based 3D photography using opacity hulls. *Pages 427–437 of: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press. 19, 34, 137

MATUSIK, W., PFISTER, H., BRAND, M. AND MCMILLAN, L. (2003). A data-driven reflectance model. *ACM Trans. Graph.*, **22**(3), 759–769. 17, 34

MAYA 2006. *http://www.autodesk.com/alias*.
AliasWavefront Maya. 100

MCALLISTER, D., LASTRA, A. AND HEIDRICH, W. (2002). Efficient Rendering of Spatial Bi-directional Reflectance Distribution Functions. *In: Graphics Hardware 2002, Eurographics / SIGGRAPH Workshop Proceedings*. 23, 31, 33, 44, 49, 54, 185, 194, 197, 221

McCool, M. D., Ang, J. and Ahmad, A. (2001). Homomorphic factorization of BRDFs for high-performance rendering. *In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM Press. 35, 45, 197

McGuire, M. and Hughes, J. F. (2004). Hardware-determined feature edges. *Pages 35–147 of: Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*. ACM Press. 66

Meseth, J., Müller, G., Sattler, M., Sarlette, R., Artusi, A., Wilkie, A., Zotti, G., Klein, R. and Purgathofer, W. (2004a)(June). High Quality Rendering of Reflectance Data. *In: Proceedings of Computer Graphics International 2004 (CGI 2004): Tutorial*. 58

Meseth, J., Müller, G. and Klein, R. (2004b). Reflectance Field based real-time, high-quality Rendering of Bidirectional Texture Functions. *Computers and Graphics*, **28**(1), 103–112. 32, 33, 38, 45, 50, 54

Mezger, J., Kimmerle, S. and Etzmuss, O. (2003). Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, **11**(2), 322–329. 247

Miller, G. (1994). Efficient Algorithms for Local and Global Accessibility Shading. *In: SIGGRAPH '94*. 88

Miller, G. S. P., Rubin, S. M. and Ponceleon, D. (1998). Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. *Pages 281–292 of: Proceedings of the 9th Eurographics Workshop on Rendering*. 18

Miller, G. S. and Hoffman, C. R. (1984). Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. *In: SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*. 48, 60, 185, 197

Mirtich, B. (1998). V-Clip: fast and robust polyhedral collision detection. *ACM Trans. Graph.*, **17**(3), 177–208. 106

Müller, G., Meseth, J. and Klein, R. (2003). Compression and Real-Time Rendering of Measured BTFs Using Local PCA. *Pages 271–280 of:* Ertl, T., Girod, B., Greiner, G., Niemann, H., Seidel, H.-P., Steinbach, E. and Westermann, R. (eds), *Vision, Modeling and Visualisation 2003*. Akademische Verlagsgesellschaft Aka GmbH, Berlin. 39, 54

MÜLLER, G., MESETH, J., SATTLER, M., SARLETTE, R. AND KLEIN, R. (2004a)(September). Acquisition, Synthesis and Rendering of Bidirectional Texture Functions. *Pages 69–94 of:* SCHLICK, C. AND PURGATHOFER, W. (eds), *Proceedings of Eurographics 2004: State of the Art Reports*. INRIA and Eurographics Association. 27

MÜLLER, G., MESETH, J. AND KLEIN, R. (2004b). Fast Environmental Lighting for Local-PCA Encoded BTFs. *In: Computer Graphics International 2004 (CGI2004)*. IEEE Computer Society Press. 53

MÜLLER, G., MESETH, J., SATTLER, M., SARLETTE, R. AND KLEIN, R. (2005a). Acquisition, Synthesis and Rendering of Bidirectional Texture Functions. *Computer Graphics Forum*, **24**(1), 83–109. 5

MÜLLER, G., BENDELS, G. H. AND KLEIN, R. (2005b). Rapid Synchronous Acquisition of Geometry and BTF for Cultural Heritage Artefacts. *Pages 13–20 of: The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*Eurographics Association, for Eurographics Association. 27

MW 2006. *http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html*.
MathWorld, Wolfram Research. 114

MWD 2005. *http://www.merriam-webster.com*.
Merriam-Webster Online Dictionary. 92, 117, 153, 191

NAVARRETE, P. AND DEL SOLAR, J. R. (2001). Eigenspace-Based Recognition of Faces: Comparisons and a New Approach. *In: Proceedings of the 11th International Conference on Image Analysis and Processing*. 141

NEIDER, J., DAVIS, T. AND WOO, M. (1997). *OpenGL Programming Guide, Second Edition*. Addison-Wesley. 67, 71

NEMO. (2003). *Finding Nemo*.
Disney, Pixar. 119

NEULANDER, I. (2003). Image-Based Diffuse Lighting Using Visibility Maps. *In: Siggraph Sketch 2003*. 90, 179

NEWMAT 2006. *http://www.robertnz.net/nm_intro.htm*.
newmat. 109

NG, R., RAMAMOORTHI, R. AND HANRAHAN, P. (2003). All-Frequency Shadows using Non-Linear Wavelet Lighting Approximation. *ACM Transactions on Graphics*, **22**(3), 376–381. 43

NICODEMUS, F. E., RICHMOND, J. C., HSIA, J. J., GINSBERG, I. W. AND LIMPERIS, T. (1970). Reflectance nomenclature and directional reflectance and emissivity. *Applied Optics*, **9**, 1474–1475. 196

NICODEMUS, F. E., RICHMOND, J. C., HSIA, J. J., GINSBERG, I. W. AND LIMPERIS, T. (1977)(October). *Geometrical Considerations and Nomenclature for Reflectance*. U.S. Department of Commerce, National Bureau of Standards. 15, 17

NISHINO, K., SATO, Y. AND IKEUCHI, K. (2001). Eigen-Texture Method: Appearance Compression and Synthesis Based on a 3D Model. *IEEE Trans. Pattern Anal. Mach. Intell.*, **23**, 1257–1265. 34, 137, 203

NV 2005. *http://developer.nvidia.com/page/home*.
NVIDIA Developer Homepage. 100

NVD 2006. *http://developer.nvidia.com/page/home*.
NVIDIA Developer Homepage. 79

OPENEXR 2006. *http://www.openexr.com/*.
OpenEXR, Industrial Light & Magic. 58

OPENGL 2005. *http://www.opengl.org*.
OpenGL. 67

OSS 2005. *http://oss.sgi.com/projects/ogl-sample/registry/*.
OpenGL® Extension Registry. 73, 74, 81, 145, 180, 207

PAJAROLA, R. AND ROSSIGNAC, J. (2000). Compressed Progressive Meshes. *IEEE Transactions on Visualization and Computer Graphics*, **6**(1), 79–93. 137

PARENT, R. (2002). *Computer Animation - Algorithms and Techniques*. Morgan Kaufmann. 92, 94, 95, 96

PERLIN, K. AND GOLDBERG, A. (1995). Real time responsive animation with personality. *In: IEEE Transactions on Visualization and Computer Graphics 1*. 120

PERLIN, K. AND GOLDBERG, A. (1996). Improv: A system for scripting interactiveactors in virtual worlds. *In: Proceedings of Siggraph 96*. 120

PHARR, M. (2004). Ambient Occlusion. *In: GDC 2004*. 90, 179

PHONG, B. T. (1975). Illumination for computer generated pictures. *Communications of the ACM*, **18**(6), 311–317. 31

PIXAR 2005. *http://www.pixar.com*.
PIXAR. 21, 99

PRESS, W., TEUKOLSKY, S., VETTERLING, W. AND FLANNERY, B. (1992). *Numerical recipes in C - The art of scientific computation*. 2nd edn. Cambridge University Press. 31, 34, 109, 121, 203

PUPPO, E. AND SCOPIGNO, R. (1997). Simplification, lod and multiresolution - principles and applications. *In: Eurographics 1997 Tutorial Notes*. Eurographics Association. 63

PURCELL, T., BUCK, I., MARK, W. AND HANRAHAN, P. (2002). Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics*, **21**, 703–712. 91, 178

RADLOFF, J. (2004). *Obtaining the Bidirectional Texture Reflectance of Real-World Surfaces by means of a Kaleidoscope*. Tech. rept. Department of Computer Science, Rhodes University. 27

RAMAMOORTHI, R. AND HANRAHAN, P. (2002). Frequency space environment map rendering. *Pages 517–526 of: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press. 36, 197, 203

RANDIMA, F. (ed). (2004). *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addison-Wesley Professional. 90, 179

REALREFLECT 2006. *http://www.realreflect.org*.
RealReflect, EU Projekt. 236

REEVES, W. T., SALESIN, D. H. AND COOK, R. L. (1987). Rendering antialiased shadows with depth maps. *Pages 283–291 of: SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. ACM Press. 82, 83

REGE, A. (2002). Occlusion (HP and NV Extensions) Occlusion. *In: Game Developers Conference 2002*. 74

ROSSIGNAC, J. (1999). Edgebreaker: Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, **5**(1), 47–61. 137, 148

ROSSIGNAC, J. (2004). *Surface simplification and 3D geometry compression; Chapter 54 in Handbook of Discrete and Computational Geometry*. 2nd edn. Editors: J. E. Goodman and J. O'Rourke. 143

ROST, R. J. (2004). *OpenGL(R) Shading Language*. Addison Wesley Longman Publishing Co., Inc. 73

SAFF, E. AND KUIJLAARS, A. (1997). Distributing many points on a sphere. *Mathematical Intelligencer*, **19**, 5–11. 183

SANDER, P. V., HOPPE, H., SNYDER, J. AND GORTLER, S. J. (2001). Discontinuity Edge Overdraw. *In: ACM Symposium on Interactive 3D Graphics 2000*. 221

SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H. AND SNYDER, J. (2000). Silhouette Clipping. *In: Proceedings of SIGGRAPH 2000*. Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH. 221

SATTLER, M., SARLETTE, R. AND KLEIN, R. (2003). Efficient and realistic visualization of cloth. *Pages 167–177 of: EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*. Eurographics Association. 5, 38, 46, 51, 54, 90, 179, 221, 222, 223, 226, 227

SATTLER, M., SARLETTE, R., ZACHMANN, G. AND KLEIN, R. (2004a). Hardware-accelerated ambient occlusion computation. *Pages 331–338 of:* GIROD, B., MAGNOR, M. AND SEIDEL, H.-P. (eds), *Proceedings of Vision, Modeling, and Visualization 2004*. Akademische Verlagsgesellschaft Aka GmbH, Berlin. 5, 74, 177

SATTLER, M., SARLETTE, R. AND KLEIN, R. (2004b)(June). Probabilistic Motion Sequence Generation. *Pages 514–517 of: Proceedings of Computer Graphics International 2004 (CGI 2004)*. IEEE Computer Society. 5, 85, 90, 119

SATTLER, M., SARLETTE, R., MÜCKEN, T. AND KLEIN, R. (2005a). Exploitation of human shadow perception for fast shadow rendering. *Pages 131–134 of: APGV 2005: Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization*. ACM. 5, 156

SATTLER, M., SARLETTE, R. AND KLEIN, R. (2005b). Simple and efficient compression of animation sequences. *Pages 209–217 of: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM Press. 5, 136

SCHNEIDER, M. (2004). Real-Time BTF Rendering. *In: Proceedings of CESCG 2004*. 47

SCHÖDL, A., SZELISKI, R., SALESIN, D. H. AND ESSA, I. (2000). Video Textures. *Pages 489–498 of:* AKELEY, K. (ed), *Siggraph 2000, Computer Graphics Proceedings*. ACM Press / ACM SIGGRAPH / Addison Wesley Longman. 119, 120, 121, 122

SEETZEN, H., HEIDRICH, W., STUERZLINGER, W., WARD, G., WHITEHEAD, L., TRENTACOSTE, M., GHOSH, A. AND VOROZCOVS, A. (2004). High dynamic range display systems. *ACM Trans. Graph.*, **23**(3), 760–768. 57

SEGAL, M., KOROBKIN, C., VAN WIDENFELT, R., FORAN, J. AND HAEBERLI, P. (1992). Fast shadows and lighting effects using texture mapping. *SIGGRAPH Comput. Graph.*, **26**(2), 249–252. 81

SHIRLEY, P. (2000). *Realistic Ray Tracing*. A K Peters. 162

SHLAFMAN, S., TAL, A. AND KATZ, S. (2002). Metamorphosis of polyhedral surfaces using decomposition. *Computer Graphics Forum*, **21**(3), 219–228. 138

SLOAN, P.-P., KAUTZ, J. AND SNYDER, J. (2002). Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *Pages 527–536 of: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press. 43, 50, 177, 179, 197

SLOAN, P.-P., LIU, X., SHUM, H.-Y. AND SNYDER, J. (2003a). Bi-Scale Radiance Transfer. *ACM Transactions on Graphics*, **22**(3), 370–375. 50, 53, 137

SLOAN, P.-P., HALL, J., HART, J. AND SNYDER, J. (2003b). Clustered Principal Components for Precomputed Radiance Transfer. *ACM Transactions on Graphics*, **22**(3), 382–391. 53

SOLER, C. AND SILLION, F. X. (1998). Fast Calculation of Soft Shadow Textures Using Convolution. *Computer Graphics*, **32**(Annual Conference Series), 321–332. 83

STAMMINGER, M. AND DRETTAKIS, G. (2002). Perspective shadow maps. *In: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press. 81, 207

STANEKER, D., BARTZ, D. AND MEISSNER, M. (2003). Improving Occlusion Query Efficiency with Occupancy Maps. *Pages 111–118 of: Proc. of Symposium on Parallel and Large Data Visualization and Graphics.* 74

STANEKER, D. (2003). An Occlusion Culling Toolkit for OpenSG PLUS. *In: Proc. of OpenSG 2003 Symposium.* 74

STAR. (1982). *Star Trek: The Wrath of Khan.*
Paramount. 93, 243

STEWART, A. J. (1999). Computing Visibility from Folded Surfaces. *Computers & Graphics*, **23**(5), 693–702. 89, 162, 179

STEWART, A. J. AND LANGER, M. S. (1997). Towards Accurate Recovery of Shape from Shading under Diffuse Lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**(9), 1020–1025. 162, 163, 176

SUTHERLAND, I. E. (1963). Sketchpad: A man-machine graphical communication system. *In: Summer Joint Computer Conference.* Spartan Book. 67

SUYKENS, F., VOM BERGE, K., LAGAE, A. AND DUTRÉ, P. (2003)(September). Interactive Rendering of Bidirectional Texture Functions. *Pages 463–472 of: Eurographics 2003.* 35, 40, 45, 54

TADAMURA, K., QIN, X., JIAO, G. AND NAKAMAE, E. (2001). Rendering optimal solar shadows with plural sunlight depth buffers. *The Visual Computer*, **17**(2), 76 – 90. 82

TAUBIN, G. AND ROSSIGNAC, J. (1999). 3D Geometry Compression. *In: Siggraph Course Notes.* 137

TAUBIN, G. AND ROSSIGNAC, J. (1998). Geometric compression through topological surgery. *ACM Transactions on Graphics*, **17**(2), 84–115. 137

TERZOPOULOS, D. AND FLEISCHER, K. (1988). Modeling inelastic deformation: viscolelasticity, plasticity, fracture. *Pages 269–278 of: SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques.* New York, NY, USA: ACM Press. 100, 103

TERZOPOULOS, D., PLATT, J., BARR, A. AND FLEISCHER, K. (1987). Elastically Deformable Models. **21**(4), 205–214. 100

TESCHNER, M., KIMMERLE, S., HEIDELBERGER, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W. AND VOLINO, P. (2004). Collision Detection for Deformable Objects. *In: Eurographics 2004 STAR 5*. The Eurographics Association. 107

TESCHNER, M., MANOCHA, D., HEIDELBERGER, B., GOVINDARAJU, N., ZACHMANN, G., MEZGER, J. AND FUHRMANN, A. (2005). Collision handling in dynamic simulation environments. *In: Eurographics 2005 Tutorial Notes T2*. The Eurographics Association. 107

TESCHNER, M., CANI, M.-P., FEDKIW, R., REDON, S., VOLINO, P. AND ZACHMANN, G. (2006). Collision Handling and its Applications. *In: Eurographics 2006 Tutorial Notes*. The Eurographics Association. 107

TOBIAS ISENBERG, BERT FREUDENBERG, N. H. S. S. T. S. (2003). A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications*, **23**(4), 28–37. 66

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B. AND SHUM, H.-Y. (2002). Synthesis of bidirectional texture functions on arbitrary surfaces. *Pages 665–672 of: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press. 50, 199

TOUMA, C. AND GOTSMAN, C. (1998)(June). Triangle Mesh Compression. *Pages 26–34 of: Graphics Interface*. 137

TOY. (1995). *Toy Story*.
PIXAR Studios. 93

TURK, G. AND LEVOY, M. (1994). Zippered Polygon Meshes from Range Images. *In: Siggraph 1994, Computer Graphics Proceedings*. 248

UNR 2006. *http://www.unrealtechnology.com*.
Unreal 3, Epic Games. 78

VAN GELDER, A. AND WILHELMS, J. (1997). An Interactive Fur Modeling Technique. *Pages 181–188 of:* DAVIS, W. A., MANTEI, M. AND KLASSEN, R. V. (eds), *Graphics Interface '97*. Canadian Human-Computer Communications Society. 220

VASILESCU, M. A. O. AND TERZOPOULOS, D. (2004). TensorTextures: multilinear image-based rendering. *ACM Trans. Graph.*, **23**(3), 336–342. 37

VOLINO, P. AND MAGNENAT-THALMANN, N. (2000). *Virtual Clothing - Theory and Practice*. Springer-Verlag, Berlin. 100

VOLINO, P., COURCHESNE, M. AND THALMANN, N. M. (1995). Versatile and efficient techniques for simulating cloth and other deformable objects. *Pages 137–144 of: SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press. 100

VOLINO, P., THALMANN, N. M., JIANHUA, S. AND THALMANN, D. (1996). An Evolving System for Simulating Clothes on Virtual Actors. *IEEE Comput. Graph. Appl.*, **16**(5), 42–51. 100

VTO 2005. *http://www.virtualtryon.de/*.
Virtual Try-On, BMBF Projekt. 5, 230, 247, 248

WACKER, M., KECKEISEN, M., KIMMERLE, S., STRASSER, W., LUCKAS, V., GROSS, C., FUHRMANN, A., SATTLER, M., SARLETTE, R. AND KLEIN, R. (2004). Virtual Try-On. *Informatik Spektrum*, **27**(6), 504–511. 5, 235

WACKER, M., KECKEISEN, M., KIMMERLE, S., STRASSER, W., LUCKAS, V., GROSS, C., FUHRMANN, A., SATTLER, M., SARLETTE, R. AND KLEIN, R. (2005). Simulation and Visualisation of Virtual Textiles for Virtual Try-On. *Special Issue of Research Journal of Textile and Apparel: Virtual Clothing Technology and Applications*, **9**(1), 37–47. 235

WACKER, M., STRASSER, W., MAGNENAT-THALMANN, N., VOLINO, P. AND THOMASZEWSKI, B. (2006). High Performance - Virtual Garment Simulation. *In: Eurographics 2006 Tutorial Notes*. The Eurographics Association. 100

WALD, I., BENTHIN, C. AND SLUSALLEK, P. (2003a). Interactive Global Illumination in Complex and Highly Occluded Environments. *Pages 74–81 of: Proceedings of the 14th Eurographics Symposium on Rendering*. Eurographics Association. 91, 178

WALD, I., PURCELL, T. J., SCHMITTLER, J., BENTHIN, C. AND SLUSALLEK, P. (2003b). Realtime Ray-Tracing and its use for Interactive Global Illumination. *In: Eurographics 2003: State of the Art Reports*. 42, 44, 91, 178

WANGER, L. (1992). The effect of shadow quality on the perception of spatial relationships in computer generated imagery. *Pages 39–42 of: SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*. New York, NY, USA: ACM Press. 156

WANGER, L. C., FERWERDA, J. A. AND GREENBERG, D. P. (1992). Perceiving Spatial Relationships in Computer-Generated Images. *IEEE Comput. Graph. Appl.*, **12**(3), 44–51, 54–58. 156

WATT, A. AND WATT, M. (1991). *Advanced animation and rendering techniques*. New York, NY, USA: ACM Press. 92

WEBER, E. H. (1834). *De Pulsu, Resorptione, Auditu et Tactu - Annotationes Anatomicae Et Physiologicae*. Koehler. 158

WEIL, J. (1986). The synthesis of cloth objects. *Pages 49–54 of: SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. ACM Press. 100, 101, 102

WHYTE, L. L. (1952). Unique Arrangement of Points on a Sphere. *Amer. Math. Monthly 59*, **59**, 606–611. 183

WILLIAMS, L. (1978). Casting curved shadows on curved surfaces. *Pages 270–274 of: SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. ACM Press. 79, 156, 178, 187, 207

WIMMER, M., SCHERZER, D. AND PURGATHOFER, W. (2004). Light Space Perspective Shadow Maps. *Pages 143–151 of:* KELLER, A. AND JENSEN, H. W. (eds), *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering 2004)*Eurographics Association, for Eurographics. 81

WONG, T.-T., HENG, P.-A., OR, S.-H. AND NG, W.-Y. (1997). Image-based Rendering with Controllable Illumination. *Pages 13–22 of: Proceedings of EGRW '97*. Springer-Verlag. 29

WOOD, D. N., AZUMA, D. I., ALDINGER, K., CURLESS, B., DUCHAMP, T., SALESIN, D. H. AND STUETZLE, W. (2000). Surface Light Fields for 3D Photography. *Pages 287–296 of:* AKELEY, K. (ed), *Siggraph 2000, Computer Graphics Proceedings*. ACM Press / ACM SIGGRAPH / Addison Wesley Longman. 18

WOOTEN, W. L. AND HODGINS, J. K. (1996). Animation of human diving. *Computer Graphics Forum*, **1**(15), 3–13. 120

WYMAN, C. AND HANSEN, C. (2003). Penumbra Maps: Approximate Soft Shadows in Real Time. *Pages 202–207 of: Proceedings of the 2003 Eurographics Symposium on Rendering*. Eurographics Association. 84, 156

Xu, Y.-Q., Chen, Y., Lin, S., Zhong, H., Wu, E., Guo, B. and Shum, H.-Y. (2001). Photorealistic rendering of knitwear using the lumislice. *Pages 391–398 of: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques.* ACM Press. 195, 221

Ying, Z., Tang, M. and Dong, J. (2002). Soft Shadow Maps for Area Light by Area Approximation. *Page 442 of: PG '02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications.* Washington, DC, USA: IEEE Computer Society. 84

Zhang, Z. (2000). A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(11), 1330–1334. 25

Zhukov, S., Iones, A. and Kronin, G. (1998). Ambient Light Illumination Model. *In: Proc. Eurographics Rendering Workshop '98.* 88, 179

Publication List

The work presented in this thesis is mainly based on the following publications of the author of this thesis:

GANSTER, B., KLEIN, R., SATTLER, M. AND SARLETTE, R. (2002). Realtime Shading of Folded Surfaces. *Pages 465–480 of:* VINCE, J., & EARNSHAW, R. (eds), *Advances in Modelling, Animation and Rendering*. Springer Verlag.

HAUTH, M., ETZMUSS, O., EBERHARDT, B., KLEIN, R., SARLETTE, R., SATTLER, M., DAUBERT, K. AND KAUTZ, J. (eds) (2002). *Cloth Animation and Rendering – Eurographics 2002 Tutorial Notes*. Vol. T3. The Eurographics Association.

SATTLER, M., SARLETTE, R. AND KLEIN, R. (2003). Efficient and realistic visualization of cloth. *Pages 167–177 of: EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*. Eurographics Association.

MESETH, J., MÜLLER, G., SATTLER, M. AND KLEIN, R. (2003) (November). BTF Rendering for Virtual Environments. *Pages 356–363 of: Proceedings of Virtual Concepts 2003.*

KAUTZ, J., SATTLER, M., SARLETTE, R., KLEIN, R. AND SEIDEL, H.-P. (2004). Decoupling BRDFs from Surface Mesostructures. *Pages 177–182 of: GI '04: Proceedings of the 2004 conference on Graphics interface*. Canadian Human-Computer Communications Society.

MESETH, J., MÜLLER, G., SATTLER, M., SARLETTE, R., ARTUSI, A., WILKIE, A., ZOTTI, G., KLEIN, R. AND PURGATHOFER, W. (2004)(June). High Quality Rendering of Reflectance Data. *In: Proceedings of Computer Graphics International 2004 (CGI 2004): Tutorial.*

SATTLER, M., SARLETTE, R. AND KLEIN, R. (2004b)(June). Probabilistic Motion Sequence Generation. *Pages 514–517 of: Proceedings of Computer Graphics International 2004 (CGI 2004).* IEEE Computer Society.

SATTLER, M., SARLETTE, R., ZACHMANN, G. AND KLEIN, R. (2004a). Hardware-accelerated ambient occlusion computation. *Pages 331–338 of:* GIROD, B., MAGNOR, M., & SEIDEL, H.-P. (eds), *Proceedings of Vision, Modeling, and Visualization 2004*. Akademische Verlagsgesellschaft Aka GmbH, Berlin.

MÜLLER, G., MESETH, J., SATTLER, M., SARLETTE, R. AND KLEIN, R. (2004) (September). Acquisition, Synthesis and Rendering of Bidirectional Texture Functions. *Pages 69–94 of:* SCHLICK, CHRISTOPHE, & PURGATHOFER, WERNER (eds), *Proceedings of Eurographics 2004: State of the Art Reports*. INRIA and Eurographics Association.

WACKER, M., KECKEISEN, M., KIMMERLE, S., STRASSER, W., LUCKAS, V., GROSS, C., FUHRMANN, A., SATTLER, M., SARLETTE, R. AND KLEIN, R.. (2004). Virtual Try-On. *Informatik Spektrum*, **27**(6), 504–511.

MÜLLER, G., MESETH, J., SATTLER, M., SARLETTE, R. AND KLEIN, R. (2005). Acquisition, Synthesis and Rendering of Bidirectional Texture Functions. *Computer Graphics Forum*, **24**(1).

SATTLER, M., SARLETTE, R. AND KLEIN, R. (2005b). Simple and efficient compression of animation sequences. *Pages 209–217 of: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/ Eurographics symposium on Computer animation*. ACM Press.

WACKER, M., KECKEISEN, M., KIMMERLE, S., STRASSER, W., LUCKAS, V., GROSS, C., FUHRMANN, A., SATTLER, M., SARLETTE, R. AND KLEIN, R. (2005). Simulation and Visualisation of Virtual Textiles for Virtual Try-On. *Special Issue of Research Journal of Textile and Apparel: Virtual Clothing Technology and Applications*, **9**(1).

SATTLER, M., SARLETTE, R., MÜCKEN, T. AND KLEIN, R. (2005a). Exploitation of human shadow perception for fast shadow rendering. *Pages 131–134 of: APGV 2005: Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization*. ACM.