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
Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms Universität 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 Kölische 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 change-
able distant direct high-dynamic-range illumination.

The human perception system is the main target of any computer graphics appli-
cation and can also be treated as part of the rendering pipeline. Therefore, opti-
mization 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 ex-
tended networks or to less powerful clients requires efficient compression sche-
mes. The second method included in this thesis in the animation field is a geo-
metry 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


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.


Diese Arbeit beschreibt die erste praktikable Methode, um BTF Daten effizient zu komprimieren, zu speichern und für Echtzeitanwendungen zum Visualisieren wieder zu dekompimierend. Die Methode basiert auf der Principal Component Analysis (PCA), die Daten nach Signifikanz ordnet. Während die PCA die entscheidenden 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.

Korrekte Verwendung 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

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.


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 Basisssequenz, 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 Rekonstruk-
tionsqualität nach der Dekomprimierung.
Abstract ............................................................. VII
Zusammenfassung ................................................... IX
Contents ............................................................ XV

1 Preface 1
1.1 Motivation ..................................................... 2
1.2 Main Contributions .......................................... 5
1.3 Thesis Overview .............................................. 6

2 Background 7
2.1 Rendering Techniques ........................................ 8
   2.1.1 Radiometry ............................................... 8
   2.1.2 Colorimetry & Photometry ............................. 11
   2.1.3 Shape ...................................................... 12
   2.1.4 Material .................................................. 14
   2.1.5 8D: BSSRDF ............................................. 15
   2.1.6 6D: BTF .................................................. 16
   2.1.7 4D: BRDF / LF / RF .................................. 17
   2.1.8 2D: Texture, Bump & Displacement maps .......... 19
   2.1.9 Data Acquisition ....................................... 21
   2.1.10 Compression ............................................ 29
   2.1.11 Fitting Analytical BRDF-Models .................... 31
   2.1.12 Linear Basis Decomposition ......................... 34
   2.1.13 Rendering .............................................. 41
   2.1.14 Solving the Rendering Equation .................... 42
   2.1.15 BTF-Rendering using Real-Time Raytracing ........ 44
   2.1.16 BTF-Rendering using Graphics Hardware .......... 44
## Contents

2.1.17 Interactive Rendering of BTFs with Point Lights 44
2.1.18 Interactive Rendering of BTFs with Distant Illumination 48
2.1.19 Hierarchy 54
2.1.20 Radiance Data 56
2.1.21 Illumination 59

2.2 Geometry Processing Techniques 62
2.2.1 Mesh Simplification 62
2.2.2 Level of Detail 63
2.2.3 Silhouette Edge Detection 64

2.3 Graphics Hardware 67
2.3.1 Overview 67
2.3.2 Architecture 68
2.3.3 Shader Programming 72
2.3.4 Occlusion Queries 73

2.4 Shadows 75
2.4.1 Shadow Volumes 76
2.4.2 Shadow Maps 79
2.4.3 Hard versus Soft Shadows 82
2.4.4 Ambient Occlusion & Self-Shadowing 85

2.5 Animation 92
2.5.1 Historical Overview 92
2.5.2 Animation Terminology 93
2.5.3 Animation Data Generation 95

2.6 Cloth Visualization 99
2.6.1 Historical Overview 100
2.6.2 Geometry Simulation 100
2.6.3 Rendering of Cloth 107

2.7 Data Analysis Techniques 109
2.7.1 Introduction 109
2.7.2 Data Fitting and Reduction 109
2.7.3 Multivariate Analysis 110
2.7.4 Clustering 112

3 Animation 117
3.1 Introduction 117
3.2 Sequence Generation 119
3.2.1 Introduction 119
3.2.2 Data Analysis 121
3.2.3 Results 128
3.3 Sequence Compression 136
3.3.1 Introduction 136

XIII
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2</td>
<td>Algorithm Overview</td>
<td>138</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Results</td>
<td>146</td>
</tr>
<tr>
<td>4</td>
<td>Shadows</td>
<td>153</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>153</td>
</tr>
<tr>
<td>4.2</td>
<td>Perception</td>
<td>155</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Introduction</td>
<td>155</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Related Work</td>
<td>156</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Experimental setup</td>
<td>156</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Experimental procedure</td>
<td>158</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Results</td>
<td>159</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Conclusions</td>
<td>160</td>
</tr>
<tr>
<td>4.3</td>
<td>Self-Shadowing: Static Case</td>
<td>162</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Introduction</td>
<td>162</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Related Work</td>
<td>162</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Algorithm Outline</td>
<td>163</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Preprocessing</td>
<td>164</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Comparison of Rendering Methods</td>
<td>166</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Realtime Rendering</td>
<td>169</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Results</td>
<td>171</td>
</tr>
<tr>
<td>4.3.8</td>
<td>Conclusions</td>
<td>176</td>
</tr>
<tr>
<td>4.4</td>
<td>Self-Shadowing: Dynamic Case</td>
<td>177</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Introduction</td>
<td>177</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Related Work</td>
<td>178</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Ambient Occlusion Calculation</td>
<td>179</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Optimizations</td>
<td>183</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Image Based Illumination</td>
<td>185</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Dynamic Geometry and Animations</td>
<td>186</td>
</tr>
<tr>
<td>4.4.7</td>
<td>Results</td>
<td>187</td>
</tr>
<tr>
<td>5</td>
<td>Material Reflection Properties</td>
<td>191</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>191</td>
</tr>
<tr>
<td>5.2</td>
<td>Related Work &amp; Introduction</td>
<td>193</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Modeling Mesostructure</td>
<td>195</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Measuring reflection properties</td>
<td>196</td>
</tr>
<tr>
<td>5.3</td>
<td>Acquisition</td>
<td>198</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Measuring and synthesizing BTF data</td>
<td>198</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Setup and Data Acquisition</td>
<td>199</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Postprocessing</td>
<td>203</td>
</tr>
<tr>
<td>5.4</td>
<td>Compression</td>
<td>203</td>
</tr>
<tr>
<td>5.5</td>
<td>Visualization</td>
<td>206</td>
</tr>
</tbody>
</table>
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5.1 Shadow Enhancements</td>
<td>207</td>
</tr>
<tr>
<td>5.5.2 Image-based illumination</td>
<td>209</td>
</tr>
<tr>
<td>5.6 Visualization Results</td>
<td>216</td>
</tr>
<tr>
<td>5.7 Silhouette Enhancements</td>
<td>220</td>
</tr>
<tr>
<td>5.7.1 Previous Work</td>
<td>220</td>
</tr>
<tr>
<td>5.7.2 Acquisition &amp; Postprocessing</td>
<td>221</td>
</tr>
<tr>
<td>5.7.3 Rendering</td>
<td>224</td>
</tr>
<tr>
<td>5.7.4 Results &amp; Conclusions</td>
<td>227</td>
</tr>
<tr>
<td>6 Applications</td>
<td>229</td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>229</td>
</tr>
<tr>
<td>6.2 Virtual Try-On</td>
<td>230</td>
</tr>
<tr>
<td>6.2.1 Project Description</td>
<td>231</td>
</tr>
<tr>
<td>6.3 Real Reflect</td>
<td>236</td>
</tr>
<tr>
<td>6.3.1 Project Description</td>
<td>237</td>
</tr>
<tr>
<td>7 Conclusions &amp; Future Work</td>
<td>241</td>
</tr>
<tr>
<td>7.1 Conclusions</td>
<td>241</td>
</tr>
<tr>
<td>7.2 Future Work</td>
<td>243</td>
</tr>
<tr>
<td>A Miscellaneous</td>
<td>245</td>
</tr>
<tr>
<td>Videos</td>
<td>246</td>
</tr>
<tr>
<td>Data Sources</td>
<td>247</td>
</tr>
<tr>
<td>List of Figures</td>
<td>250</td>
</tr>
<tr>
<td>List of Tables</td>
<td>254</td>
</tr>
<tr>
<td>References</td>
<td>255</td>
</tr>
</tbody>
</table>
CHAPTER 1

Preface
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 so-
motivation

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.
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
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.

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 $\omega_i$ with the wavelength $\lambda_i$ at the time $t_i$. The surface is hit at the point $x_i$ with the surface normal $n_i$. The direction $\omega_i$ is defined by two angles, namely $\theta_i$ and $\phi_i$. 

![Fig. 2.1: General light-matter interaction.](image)
and $\phi_i$. The latter is attached to a local coordinate system vector $t_i$.

After traveling through the matter, the beam leaves the surface at the time $t_0$ at the outgoing point $x_o$ into the direction $\omega_o$. It might have changed the wavelength to $\lambda_o$. Similar to the entrance direction, the direction is defined by the two angles $\theta_o$ and $\phi_o$ with the local coordinate vector $t_o$. 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].

<table>
<thead>
<tr>
<th>symbol</th>
<th>radiant term</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nu$</td>
<td>frequency</td>
<td>s$^{-1}$</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>wavelength</td>
<td>m</td>
</tr>
<tr>
<td>$E_{ph}$</td>
<td>photon energy</td>
<td>J</td>
</tr>
<tr>
<td>$Q_e$</td>
<td>radiant energy</td>
<td>J</td>
</tr>
<tr>
<td>$\Phi_e$</td>
<td>radiant flux</td>
<td>W</td>
</tr>
<tr>
<td>$E_i$</td>
<td>irradiance (incident)</td>
<td>$W m^{-2}$</td>
</tr>
<tr>
<td>$M_e$</td>
<td>radiant exitance (outgoing)</td>
<td>$W m^{-2}$</td>
</tr>
<tr>
<td>$I_e$</td>
<td>radiant intensity</td>
<td>$W sr^{-1}$</td>
</tr>
<tr>
<td>$L_e$</td>
<td>radiance</td>
<td>$W m^{-2} sr^{-1}$</td>
</tr>
</tbody>
</table>

Tab. 2.1: Important radiant terms.

<table>
<thead>
<tr>
<th>symbol</th>
<th>term</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>Planck’s constant</td>
<td>$6.626 \times 10^{-34} Js$</td>
</tr>
<tr>
<td>$c_0$</td>
<td>speed of light</td>
<td>299 792 458 m$s^{-1}$</td>
</tr>
</tbody>
</table>

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 $c$:

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

(2.1)
and a certain amount of energy:

\[ E_{ph} = h\nu = \frac{hc}{\lambda} \]  \hspace{1cm} (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:

\[ \Phi_e = \frac{dQ_e}{dt} \]  \hspace{1cm} (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 \( x \) is divided into the

irradiance (incident):

\[ E_e(x) = \frac{d\Phi_e}{dA} = \int_{2\pi sr} L_e \cdot \cos\theta \cdot d\Omega \]  \hspace{1cm} (2.4)

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

\[ M_e(x) = \frac{d\Phi_e}{dA} = \int_{2\pi sr} L_e \cdot \cos\theta \cdot d\Omega \]  \hspace{1cm} (2.5)

The solid angle \( \Omega \) 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:

\[ d\Omega = \frac{dA}{r^2} \]  \hspace{1cm} (2.6)

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

\[ I_e(\vec{\omega}) = \frac{d\Phi_e}{d\Omega} \]  \hspace{1cm} (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 \( \theta \) defines the surface normal at point \( x \):

\[
L_e(x, \vec{\omega}) = \frac{d^2\Phi_e}{dA \cdot \cos\theta \cdot d\Omega} = \frac{dI_e}{dA \cdot \cos\theta}
\]  \hspace{1cm} (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 $V(\lambda)$:

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

(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.](image)

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.](image)

![Fig. 2.5: Triangle setup.](image)

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.

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

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 \( (\lambda_i = \lambda_o) \), i.e. no fluorescence;
- wavelength is not continuous, but discretized into red, green and blue bands \( (\lambda \rightarrow \lambda_{RGB}) \);
2.1. Rendering Techniques

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.

![Diagram of reflectance functions]

**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 ($\sigma_a$: absorption cross section and $\sigma'_s$: 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.](image)

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.
2.1. Rendering Techniques

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

Fig. 2.9: BRDF representation for an infinitesimal surface element.
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.](image)

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.
Chapter 2. Background

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.

![Bump mapping with disturbed per-pixel normals.](image)

**Fig. 2.12:** Bump mapping with disturbed per-pixel normals.
2.1. Rendering Techniques

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 than 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).

![Displacement mapping](image)

*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.
Chapter 2. Background

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 $S$:

\[
\text{BTF}_{rgb}(x, \theta_i, \phi_i, \theta_r, \phi_r) := \int_S \text{BSSRDF}_{rgb}(x_i, x, \theta_i, \phi_i, \theta_r, \phi_r) \, dx_i
\]

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
2.1. Rendering Techniques

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 $10 \times 10 \text{ cm}^2$. 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.

---

**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.
Chapter 2. Background

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.

<table>
<thead>
<tr>
<th>$\theta$ [$^\circ$]</th>
<th>$\Delta \phi$ [$^\circ$]</th>
<th>No. of images</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$-^*$</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>60</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>12</td>
</tr>
<tr>
<td>45</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>60</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>75</td>
<td>15</td>
<td>24</td>
</tr>
</tbody>
</table>

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

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 $4500 \times 3000$ 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.
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.
Chapter 2. Background

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 \((256 \times 256\) pixels in size for probes in the database, up to about \(800 \times 800\) in principle). A final dataset with \(256 \times 256\) 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\textsuperscript{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 $\phi$ the covered range of the polar angle $\theta$ is restricted by the size of the mirror. For example for $\phi = \pi$ the presented prototype can capture polar angles only up to $22.6^\circ$.

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

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

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]):

$$\{B_x\}_{x \in I \subset \mathbb{N}^2}$$
Chapter 2. Background

**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 \((n \cdot I)\) between incident direction and surface normal and strong effects from meso-scale shadowing and
2.1. Rendering Techniques

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

\[ s(v, l) = \left( v^t \cdot M \cdot l \right)^n \]  

(2.10)

with \( v \) and \( l \) denoting local view and light direction respectively, while the general 3 \( \times \) 3 matrix \( M \) 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 \( D \), since fitting and rendering efforts are significantly reduced without major loss in rendering quality. Thus, they use the following, spatially varying lobes:

\[ s_x(v, l) = \left( v^t \cdot D_x \cdot l \right)^{n_x} \]  

(2.11)

This results in the following BTF approximation:

\[ BTF(x, v, l) \approx \rho_{d,x} + \sum_{j=1}^{k} \rho_{s,x,j} \cdot s_{x,j}(v, l) \]  

(2.12)

where \( \rho_d \) and \( \rho_s \) denote diffuse and specular albedo (specified as RGB values) and \( k \) is the number of lobes.
Chapter 2. Background

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 $T_x(v)$ modeling occlusion. Following their proposal, the BTF is evaluated as follows:

$$\text{BTF}(x, v, l) \approx T_x(v) \cdot \left( \rho_{d,x} + \sum_{j=1}^{k} s_{x,j}(v, l) \right). \tag{2.13}$$

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:

$$\text{BTF}(x, v, l) \approx \sum_{v \in \mathcal{N}(v)} w_{x,v} \cdot RF_{x,v}(l) \tag{2.14}$$

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

\(RF_{x,v}\) 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:

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

with \(s_{x,v}\) 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:

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

Here, \(a\) denotes the coefficients of the polynomial, \(s_{x,v}\) is defined as in equation 2.15 and \(w_l\) denotes interpolation weights for the contributions of the nearest light directions \(N(l)\) 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
can be stated as follows: Given the 4-dimensional BRDF $B_x$, find a factorization into a set of 2-dimensional functions:

$$B_x(v, l) \approx \sum_{j} c g_{x,j}(\pi_1(v, l))h_{x,j}(\pi_2(v, l))$$

(2.17)

The functions $\pi_1$ and $\pi_2$ are projection functions which map the 4D-dimensional BRDF parameters $(v, l)$ 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 $g_{x,j}$ and $h_{x,j}$ 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:

$$B_x(v, l) \approx \prod_{j} c p_{x,j}(\pi_j(v, l))$$

(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 $\pi_j$ 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:
Chapter 2. Background

\[ B_x(v, l) \approx \prod_j^{d} \sum_k^{c_j} P_{x,j,k}(\pi_{j,1}(v, l))Q_{x,j,k}(\pi_{j,2}(v, l)). \]  \hspace{1cm} (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 \(|M| \cdot |I|\) matrix

\[ X = \left( B_{x_0}, B_{x_1}, \ldots , B_{x_{|I|}} \right). \]

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

\[ \text{BTF}(x, v, l) \approx \sum_j^c g_j(x)h_j(v, l) \] \hspace{1cm} (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
2.1. Rendering Techniques

![Graph showing RMS-error of the full BTF-matrix factorization depending on the number of terms $c$.](image)

There is a direct correspondence between the magnitude and decrease of error and the BTF-complexity.

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
Chapter 2. Background

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:

$$X_{v_i} := \left( T_{(v_i,l_0)}, T_{(v_i,l_1)}, \ldots, T_{(v_i,l_{M_{v_i}})} \right)$$

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

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

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 $M_{v_i} = 81$ 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
2.1. Rendering Techniques

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:

\[
\text{BTF}(x, v, l) \approx \sum_{j} g_{k(x),j}(x)h_{k(x),j}(v, l)
\]  

The operator \(k(x)\) is the cluster index look-up given a position \(x\). 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. \(M_{v_i} = 81\)) used in per-view factorization.

<table>
<thead>
<tr>
<th>original</th>
<th>FMF</th>
<th>PCMF</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Original" /></td>
<td><img src="image2.png" alt="FMF" /></td>
<td><img src="image3.png" alt="PCMF" /></td>
</tr>
<tr>
<td><img src="image4.png" alt="Difference" /></td>
<td><img src="image5.png" alt="Difference" /></td>
<td><img src="image6.png" alt="Difference" /></td>
</tr>
</tbody>
</table>

**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.
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 \( x \) (neglecting emissive effects):

\[
L_r(x, v) = \int_{\Omega_I} \rho_x(v, l) \cdot L_i(x, l) (n_x \cdot l) \, dl \quad (2.23)
\]

Here, \( \rho_x \) denotes the BRDF, \( L_i \) denotes incoming radiance, \( n_x \) is the surface normal and \( \Omega_I \) is the hemisphere over \( x \).

Including measured BTFs into the rendering equation 2.23 is very simple:

\[
L_r(x, v) = \int_{\Omega_I} BTF(x, v, l) \cdot L_i(x, l) (n_x \cdot l) \, dl \quad (2.24)
\]

Now the measured BRDF at point \( x \) is simply looked up from the BTF. It is assumed, that a mapping from the 3D-surface to the 2D spatial texture domain already
Chapter 2. Background

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 \( \Lambda = \{ I_j \} \) of point or directional light-sources and discarding general interreflections (i.e. the recursion in equation 2.23). For notational simplicity the term \( I_j \) 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 \( x \), the local coordinates \( \tilde{l} \) are also needed:

\[
L_r(x, v) \approx \sum_{j} \rho_x(v, \tilde{l}_j) G(x, I_j)(n_x \cdot I_j) I_j \tag{2.25}
\]
2.1. Rendering Techniques

The geometry term $G(x, l_j)$ 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:

$$L_r(x, v) \approx \sum_{j} |\Lambda| BTF(x, v, \hat{l}_j)G(x, l_j)(n_x \cdot l_j)l_j$$  \hspace{1cm} (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 $x$ can be removed:

$$L_r(x, v) \approx \int_{\Omega_i} \rho_x(v, l)L_i(1)(n_x \cdot l) \, dl$$  \hspace{1cm} (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 $x$, the local view direction $v$ and in the case of point light sources the local light direction $l$.

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 $\tau_i$ 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-
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 textures 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 $h_{v,j}(x)$ with the appropriate weights $g_{v,j}(l)$ 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.](image)

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

$$\text{BTF}(x, v, l) \approx \sum_{\hat{l} \in N(l)} \tau_{\hat{l}} \sum_{j=1}^{c} g_{v,j}(\hat{l}) h_{v,j}(x)$$

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

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

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

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

**Full BTF-Matrix Factorization**

To evaluate equation 2.20 the factors \( g_j(x) \) and \( h_j(v, l) \) have to be evaluated, whereas the factors \( g_j(x) \) can be stored as simple 2D-textures and the factors \( h_j(v, l) \) 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 \( h_j(v, l) \) 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 \( h_j(v, l) \) 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 \( h_{k(x),j}(v, l) \) 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 \( g_j(x) \), which can be stored as floating-point values since they require no interpolation, have to be divided by the corresponding scaling...
Chapter 2. Background

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 ($\mathbf{BTF}(x, v, l) = \rho_d(x)$), equation 2.27 reduces to:

$$L_r(x, v) = \int_{\Omega_i} \rho_d(x)L_i(l)(n_x \cdot 1) \, dl$$

$$= \rho_d(x) \int_{\Omega_i} L_i(l)(n_x \cdot 1) \, dl$$

$$= \rho_d(x)D(n_x). \quad (2.31)$$

The prefiltered environment map $D(x, n_x)$ 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:
where \( p(x, v) \) denotes the peak direction, i.e. the light direction with maximum influence on \( L_r(x, v) \). The accuracy of this approximation increases with the specularity of the spatially varying ABRDFs. With this assumption, the specular prefiltered environment \( S(x, v) \) 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 \( D_x(n_x) \) and a specular one, which is computed based on the ideas of Kautz and McCool [Kautz & McCool 2000]. Note that whenever \( x \) is written as index, it refers to instances discretely sampled in the spatial domain.

First, the peak direction \( p_x(v) = v^t \cdot D_x \) of each lobe is computed. Then the specular illumination part (for ease of notation, a 1-lobe approximation is assumed) is rewritten as follows:

\[
L_{r,s}(x, v) \approx \int_{\Omega_i} \rho_{s,x} \cdot (p_x(v) \cdot 1)^{n_x} L_i(1)(n_x \cdot 1) \, dl
\]

\[
\approx \rho_{s,x} \cdot (n_x \cdot p_x(v)) \int_{\Omega_i} (p_x(v) \cdot 1)^{n_x} L_i(1) \, dl
\]

\[
\approx \rho_{s,x} \cdot (n_x \cdot p_x(v)) ||p_x(v)||^{n_x} S(p_x(v), n_x) \tag{2.34}
\]

with

\[
S(p_x(v), n_x) = \int_{\Omega_i} \left( \frac{p_x(v)}{||p_x(v)||} \cdot 1 \right)^{n_x} L_i(1) \, dl \tag{2.35}
\]

\( S(p, n) \) denotes the specular prefiltered environment map.

Evaluating Approximation 2.34 can efficiently be done using fragment shaders by first computing \( p_x(v) \), 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 $\hat{S}(p, n)$ 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 $B(x, v)$ is accomplished by projecting the BTF onto the first $c$ elements of the SH basis $\{Y_j\}_{j \in \mathbb{N}}$:

$$B(x, v)_j = \int_{\Omega_i} BTF(x, v, l) Y_j(l) \, dl$$

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
2.1. Rendering Techniques

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 multiplied 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 $64^3 \times 25$ 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

$$L_r(x, v) \approx \sum_{\hat{l} \in H_i} \rho_x(v, \hat{l}) H_{i,x}(\hat{l})(n_x \cdot \hat{l})$$

where $H_{i,x}$ denotes the discretized hemicube. $H_{i,x}(\hat{l})$ returns the color in the hemicube over $x$ 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 $H_{i,x}$ stores a discretization of the hemisphere at the vertex $x$. 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:
Chapter 2. Background

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:

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

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

\[
L_r(\mathbf{x}, \mathbf{v}) \approx \sum_{j=1}^c \gamma_{\mathbf{v},j} h_{\mathbf{v},j}(\mathbf{x})
\]  (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 \( \gamma_{\mathbf{v},j} \) can be introduced:

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

This vector has to 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 stan-
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].
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 Section 2.1.6) 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.
Chapter 2. Background

2.1.2 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.

![Real world radiance data example.](image)

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>G</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varnothing$</td>
<td>1.3</td>
<td>1.2</td>
<td>1.0</td>
</tr>
<tr>
<td>sun</td>
<td>2648.0</td>
<td>3640.0</td>
<td>2280.0</td>
</tr>
</tbody>
</table>

**Fig. 2.26:** Real world radiance data example.

**Tab. 2.5:** Real world radiance data example values.

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

$$D = \log \frac{I_{\text{max}}}{I_{\text{min}}}$$

(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,
2.1. Rendering Techniques

![Radiance data processing pipeline](image)

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
Chapter 2. Background

- 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 milestone 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.
2.1. Rendering Techniques

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

\[ c = c_i^{1/\gamma} \]  

(2.42)

where \( c_i \) is the input color and \( \gamma \) 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:

\textit{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:

\textit{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
Chapter 2. Background

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 visually 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 approximatively 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.

![Figure 2.28: High-dynamic range assembly with LDR images.](image)

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.](image)

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].
Chapter 2. Background

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 $M_{n-1}$ containing less geometry than the original mesh $M_n$, while resembling a similar appearance. This procedure is carried out until the coarsest mesh $M_0$.

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 $uv$ of the mesh $M_n$ is collapsed into the point $v$, thereby removing the shown triangles $A$ and $B$ and resulting in mesh $M_{n-1}$.

The vertex merge operator is shown in figure 2.31. The two vertices $v_1$ and $v_2$ are merged into a single new one $v_3$.

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...
2.2 Geometry Processing Techniques

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 exist, 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 Hausdorff, mathematician
Chapter 2. Background

The bottom row shows the common usage of several LODs. Assuming that the objects move away from the user, it is possible to use increasing LOD without losing 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 back-facing 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.

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.
Chapter 2. Background

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
2.3. Graphics Hardware

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 tessellate 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 clear 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.

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 **Red**, **Green**, **Blue** and **Alpha**. 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).
Chapter 2. Background

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.

```glsl
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
2.3. Graphics Hardware

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 \texttt{ARB}\_\texttt{fragment}\_\texttt{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 \texttt{ARB}\_\texttt{fragment}\_\texttt{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.

\begin{lstlisting}[language=GLSL]
1 //ARBfp1.0
2 # texturing with HDR environment
3 OUTPUT output=result.color;
4 TEMP component00;
5 TEX component00,fragment.texcoord[0],texture[0],2D;
6 MUL component00,fragment.texcoord[1],component00;
7 MUL output,fragment.texcoord[2].x,component00;
8 END
\end{lstlisting}

\textbf{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*. 
Chapter 2. Background

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.
2.4. Shadows

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,
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.
2.4. Shadows

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 $La_1$ and $Lb_1$ 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. 2.40: Shadow volumes in side view; z-fail algorithm.*
Chapter 2. Background

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 $b_2$ is larger than the depth value stored at $b$ in the depth map $(Lb_2)$, therefore $b_2$ is in shadow.

![Fig. 2.43: Shadow map algorithm examples; Left: light view; Center: depth map; Right: eye view.](image)

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 [Lo- kovic & 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).
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].


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.](image)

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.](image)

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-
2.4. Shadows

![Diagram](image)

**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 $M$
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 $O(N \times M)$. 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,
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 hemi-cubes [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


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].


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.

<table>
<thead>
<tr>
<th>property</th>
<th>characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>appearance</td>
<td>· 2d-drawing, cartoon-like</td>
</tr>
<tr>
<td></td>
<td>· 3d wireframe</td>
</tr>
<tr>
<td></td>
<td>· body surface properties</td>
</tr>
<tr>
<td></td>
<td>· muscles, adipose tissue</td>
</tr>
<tr>
<td></td>
<td>· biomechanics</td>
</tr>
<tr>
<td></td>
<td>· cloth, body equipment</td>
</tr>
<tr>
<td>functions</td>
<td>· included bones model</td>
</tr>
<tr>
<td></td>
<td>· constraint angles</td>
</tr>
<tr>
<td></td>
<td>· constraint forces</td>
</tr>
<tr>
<td></td>
<td>· physical condition</td>
</tr>
<tr>
<td></td>
<td>· physical capabilities</td>
</tr>
<tr>
<td></td>
<td>· cognitive capabilities</td>
</tr>
<tr>
<td>individuality</td>
<td>· gender &amp; age</td>
</tr>
<tr>
<td></td>
<td>· generic character</td>
</tr>
<tr>
<td></td>
<td>· hand-made character</td>
</tr>
<tr>
<td></td>
<td>· racial characteristics</td>
</tr>
<tr>
<td></td>
<td>· cultural characteristics</td>
</tr>
<tr>
<td></td>
<td>· psychological characteristics</td>
</tr>
<tr>
<td></td>
<td>· personality</td>
</tr>
</tbody>
</table>

**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
Chapter 2. Background

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.
2.5. Animation

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.
### Chapter 2. Background

<table>
<thead>
<tr>
<th>system</th>
<th>advantages</th>
<th>disadvantages</th>
<th>sensors</th>
</tr>
</thead>
</table>
| **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*. 

---

98
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.

Figure 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].


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
2.6. Cloth Visualization

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

- **Cosserat models**
  based 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

\[
y = c - \left( a \cdot \cosh \left( \frac{x - b}{a} \right) \right)
\] (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
2.6. Cloth Visualization

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
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.
2.6. Cloth Visualization

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 [Mir-tich 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.
Chapter 2. Background

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

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

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.
Chapter 2. Background

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 an efficient technique to solve most linear least-squares problems (see also 2.7.4) and based on the following theorem:

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

\[
X = U \ D \ V^T
\]  

(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.
2.7. Data Analysis Techniques

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, \( x_1, x_2, \ldots, x_n \) into another set of n-dimensional vectors \( y_1, y_2, \ldots, y_n \). 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 \( x \), first the sample variance (covariance) \( C \) is used:

\[
C = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T
\]  

(2.46)

with

\[
\bar{x} = \frac{1}{n} \sum_i x_i
\]

(2.47)

The eigenvalues of \( C \) are:

\[
\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_k \geq 0
\]

(2.48)

with the corresponding eigenvectors or principal axes:
Chapter 2. Background

\[ e_1, e_2, e_3 \ldots e_k \]  \hspace{1cm} (2.49)

or

\[ \lambda e = C e = \sum_i \langle x_i, e \rangle x_i \]  \hspace{1cm} (2.50)

Therefore, the eigenvectors can be written as a linear combinations of the input data multiplied with a \textit{weighting factor} \( \alpha \):

\[ e_i = \sum_i \alpha_i x_i \]  \hspace{1cm} (2.51)

If the eigenvalues are sorted according equation 2.48 the first \( k \) eigenvectors or \textit{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:

\[ \bar{\lambda} = \frac{1}{n} \sum \lambda_i \]  \hspace{1cm} (2.52)

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

\[ \lambda_i < \bar{\lambda} \]  \hspace{1cm} (2.53)

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

2.7.4 Clustering

Clustering is the process of identifying groups, called \textit{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 criterion 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.

![Diagram of clustering schemes](image)

**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:

$$E = \sum_{j=1}^{K} \sum_{n \in S_j} |x_n - c_j|^2$$  \hspace{1cm} (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 straightforward 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:

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

where \( x \) is the original and \( \hat{x}_j \) 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.