Doctoral Thesis

A multiresolution representation for light field acquisition and processing

Author(s):
Lütolf, Reto

Publication Date:
2004

Permanent Link:
https://doi.org/10.3929/ethz-a-004937503

Rights / License:
In Copyright - Non-Commercial Use Permitted
A multiresolution Representation for Light Field Acquisition and Processing

A dissertation submitted to the
Swiss Federal Institute of Technology, ETH, Zurich

for the degree of
Doctor of Technical Sciences

presented by
RETO LÜTOLF
Dipl. Informatik–Ing. ETH
Swiss Federal Institute of Technology, ETH, Zurich
born February 7, 1973
citizen of Horw, Lucerne, Switzerland

accepted on the recommendation of
Prof. M. H. Gross, examiner
Prof. B. Schiele, co-examiner

2004
ABSTRACT

Image-based rendering has been a very active field of research during the past years and thus attracted a lot of attention. The main focus was mainly put on efficiently computable reconstruction algorithms, i.e. rendering. Clearly, the design of an underlying representation always was an important issue, too, since it ought to optimally support the reconstruction method. However, well-fitting representations in particular have been addressed only scarce in the context of compression of image-based models. Moreover, even less research has been conducted in the scope of suitable representations for fast data acquisition while still enabling efficient rendering, of course. The construction of such a dynamic light field representation plus a set of appropriate operators for rapid data acquisition and processing is the main focus of this thesis.

Following a short introduction and a concise survey of relevant image-based methods an overview of the main system components developed in the course of this thesis along with their interaction is given. The hierarchical representation for light field data forms the cardinal part of the core component. Its construction results in a multi-dimensional wavelet-transformed representation, a so-called extended multiresolution analysis that also includes a hierarchical scattered data interpolation. Because this first construction is based on a limited set of input images the core continuously acquires new, i.e. previously unused image data afterwards in order to selectively refine the hierarchical representation. A procedure called oracle mechanism automatically decides on the acceptance of new input imagery individually by comparing it to the corresponding reconstruction computed from the wavelet pyramid. Accepted images are incrementally inserted, rejected images are discarded.

The set of operators required for such interaction exploits the spatial localization of wavelets by computing both image decompositions, used for the incremental update scheme, and image reconstructions, used for rendering, locally in the wavelet domain. Consequently, a costly global inverse transform of the hierarchical representation as a whole is not required by any of these local operators, neither for image data insertion nor for data read-out. Analyses of the visual and the computational performance demonstrate the effectiveness of this concept.

The dynamic representation along with the local decomposition and local rendering operators thus constitutes a powerful framework for light field acquisition and processing. Moreover, being based on wavelet transform, the representation is also amenable to compression by further exploiting the wavelets' well-known potential for high compression gains. The design of a lossy multi-stage compression scheme based on coefficient thresholding enables effective in-core and out-of-core compression. A custom-tailored data structure gives access to compressed in-core data, neither invalidating the dynamic setting nor restricting the functionality of the set of local operators in any way. On the other hand, the file format used for permanent storage of compressed in-core data on disk firstly further reduces data set size and secondly allows for progressive read back.

The thesis concludes with a broad evaluation of the unified representation and the set of local operators using real-world video imagery as input material.
Seite Leer / Blank leaf
ZUSAMMENFASSUNG


Die dafür notwendigen Operatoren nutzen die gute örtliche Lokalisierbarkeit der Wavelets aus, indem die notwendigen Berechnungen direkt und nur lokal im Raum der Wavelets durchgeführt werden—and zwar sowohl für die Zerlegung neuer Bilder für inkrementelle Aktualisierungen als auch für die Bildrekonstruktion. Eine teure, globale Rücktransformation wird daher gar nie notwendig. Verschiedene Analysen hinsichtlich visueller Qualität und Effizienz unterstreichen die Leistungsfähigkeit dieses Konzeptes.


Ausgedehnte Evaluationen der Repräsentation sowie der beiden lokalen Operatoren anhand realer Videodaten als Eingabematerial schlossen die Arbeit ab.
Seite Leer / Blank leaf
ACKNOWLEDGEMENT

At this point, I would like to take the opportunity to express my gratitude to all the people who generously offered me their support, their time and energy to help me completing this thesis.

First of all, my thankfulness goes to my advisor, Prof. Markus Gross, for letting me be part of the Computer Graphics Laboratory at ETH Zurich as one of his Ph.D. students and giving me the opportunity to work in this challenging field of computer graphics. It was most exciting to witness Markus' enthusiasm and dedication to establish this group led by him as one of the world's top research labs within just a few years. Similarly, my thanks go to my co-advisor, Prof. Bernt Schiele, especially for his guidance, his patience and all his solid and perspicacious comments. It is very clear to all of us that this work would not have been finished without Markus' and Bernt's generous assistance and tolerating willingness to follow the taken track all through the rough mountains.

In addition, many thanks go to all my fellow Ph.D. students at the Computer Graphics Laboratory and at the Department of Computer Science for truly helping me along the way. Special thanks go to Lars Lali Lippert, Oliver Staadt, Martin Tinu Roth, Tom TC Sprenger, Andreas Hubeli, Florian Nussberger, Matthias Mats Zwicker, Mark Pauly, Edouard Lamboray, Stephan Würmlin, Tim Weyrich, Christoph Niederberger and Remo Ziegler—and not to forget Daniel Bielser whom i shared my office with from the very beginning on.

Further, I would like to thank all the diploma and semester students who contributed to this work, namely David Kleiner, Stephan Würmlin, Jürg Senn and Patrick Grawehr. Additionally, it was a great pleasure to work with Remo Ziegler, Pascal Glardon, Pascal Kurtansky and again Stephan Würmlin on various other projects in the field of computer graphics.
Seite Leer / Blank leaf
CONTENTS

1 Introduction ................................................................. 1
  1.1 Motivation & Goals .................................................. 2
    1.1.1 Acquisition Scheme .......................................... 2
    1.1.2 Suitable Representation ...................................... 3
    1.1.3 Control of Memory Consumption ............................. 4
  1.2 Contribution ......................................................... 4
  1.3 Outline & Organization ............................................. 6

2 Image-based Methods ................................................... 9
  2.1 Overview ............................................................. 13
  2.2 Light Field & Lumigraph Rendering ............................... 17
    2.2.1 Characterization ............................................... 17
    2.2.2 Similarities .................................................... 18
    2.2.3 Differences .................................................... 20
    2.2.4 Follow-up Work ............................................... 25
  2.3 State of the Art ..................................................... 29
    2.3.1 Acquisition .................................................... 29
    2.3.2 Parameterization .............................................. 33
    2.3.3 Compression ................................................... 37
    2.3.4 Wavelets ....................................................... 40

3 System Overview ....................................................... 43
  3.1 Acquisition Component ............................................. 43
  3.2 Parameterization Component ...................................... 45
  3.3 System’s Core Component .......................................... 46
    3.3.1 Approximation Mode .......................................... 47
    3.3.2 Oracle Mode ................................................... 47
    3.3.3 Mode Switching ............................................... 49
  3.4 Viewing Component ................................................ 49

4 Data Representation ................................................... 51
  4.1 Necessities ........................................................ 51
  4.2 Approximation of Functions ...................................... 53
4.3 Wavelets ................................................................. 56
  4.3.1 Overview ............................................................ 56
  4.3.2 Multiresolution Analysis (MRA) ................................. 57
  4.3.3 Properties of Wavelets .............................................. 59
  4.3.4 Orthogonality ......................................................... 61
  4.3.5 Multidimensional Bases ............................................. 63
  4.3.6 Coefficient Thresholding .......................................... 64

4.4 Hierarchical Representations for IBMR Methods ...................... 66

4.5 Scattered Data Interpolation Schemes .................................. 67
  4.5.1 Shepard’s Methods ................................................... 69
  4.5.2 Burt’s Hierarchical Polynomial Fit Filters ....................... 70
  4.5.3 Gortler’s Splat–Pull–Push Interpolation Filter .................. 71

4.6 Construction of the eMRA .............................................. 75

4.7 Incremental Updates ................................................... 79

4.8 Oracle Mechanism ...................................................... 79

4.9 Visual Performance ..................................................... 80

5 Local Operators .............................................................. 85
  5.1 Concept ................................................................. 85
     5.1.1 Fundamental Idea .................................................. 86
     5.1.2 Viable Strategies .................................................. 86
     5.1.3 Implementation .................................................... 87
     5.1.4 Example in a 2D Setting .......................................... 89
     5.1.5 Use of Parameterization .......................................... 91
     5.1.6 Comparable Work .................................................. 92

  5.2 Local Reconstruction (Rendering) ..................................... 93
     5.2.1 2-dimensional Case .............................................. 93
     5.2.2 4-dimensional Case .............................................. 94
     5.2.3 Caching Mechanism .............................................. 96

  5.3 Local Projection (Incremental Update) ................................ 98
     5.3.1 2-dimensional Case .............................................. 98
     5.3.2 4-dimensional Case .............................................. 100
     5.3.3 Buffer Clearing Strategy ....................................... 101

  5.4 Computational Performance .......................................... 102
     5.4.1 First-level Transform ........................................... 102
     5.4.2 Measured Performance ........................................... 103

6 Implementation Details ................................................... 109
  6.1 Parameterization ...................................................... 109
     6.1.1 Cylinder–plane Setting ......................................... 110
     6.1.2 Sphere–plane Setting ............................................ 111
     6.1.3 Ray–based vs. Sliced Edit Operations .......................... 112
Seite Leer / Blank leaf
FIGURES

Figure 1.1 The Digital Michelangelo Project: Acquisition of a large-scale light field. . . . 3
Figure 2.1 Examples of densely sampled light fields. .................................................. 10
Figure 2.2 Classification of image-based models via rendering spectrum. .................. 11
Figure 2.3 Classification of image-based models via input processing. ....................... 13
Figure 2.4 Golden Gate Fly-over, a moviemap of the San Francisco Bay Area. ........+. 14
Figure 2.5 Parameters of the Plenoptic function, reduced to 5 dimensions. ................. 16
Figure 2.6 The light slab, a 2-plane Light field parameterization. ............................. 20
Figure 2.7 Setup for image acquisition. ................................................................... 21
Figure 2.8 Depth-correction extension in the Lumigraph system (1). ......................... 23
Figure 2.9 Depth-correction extension in the Lumigraph system (2). ......................... 24
Figure 2.10 Possible settings for the parameterization of oriented lines in 3D. .......... 34
Figure 2.11 Polar parameterization for oriented lines in 3D space. ............................. 35
Figure 3.1 Conceptual overview of the prototype system. ......................................... 43
Figure 3.2 Close-up of the acquisition component. .................................................... 44
Figure 3.3 Close-up of the parameterization component. ......................................... 45
Figure 3.4 Close-up of the core component, approximation mode. ......................... 47
Figure 3.5 Close-up of the core component, oracle mode. ........................................ 48
Figure 3.6 Close-up of the viewing component. ......................................................... 49
Figure 4.1 2D wavelet transform following the standard decomposition scheme. .... 64
Figure 4.2 2D wavelet transform following the non-standard decomposition scheme. . 65
Figure 4.3 Wavelet-compressed images using coefficient threshold filtering. .......... 67
Figure 4.4 Result of the HSDI scheme from 50% of known data. ............................... 73
Figure 4.5 Result of the HSDI scheme from 20% of known data. ............................... 74
Figure 4.6 Overview of the eMRA at construction time. ............................................ 75
Figure 4.7 Operator setting used for the construction of the eMRA. ......................... 77
Figure 4.8 The system's different modes demonstrated using the Truck light field. ....... 80
Figure 4.9 Incremental update operation using the Falls light field. ......................... 82
Figure 5.1 Wavelet filter support areas for a local point update in a 2D setting. ......... 90
Figure 5.2 Individual processing steps for a local point reconstruction. ..................... 93
Figure 5.3 Example (1) of a local point reconstruction in 2D. .................................... 94
Figure 5.4 Example (2) of a local point reconstruction in 2D. .................................... 95
Figure 5.5 Filtering steps for a local reconstruction operation of a single plane in 4D. . 95
Figure 5.6 Squared cache area in the parameter space. ............................................. 97
Figure 5.7 Filtering steps for a local projection operation of a single line in 2D. ........... 99
Figure 5.8 Filtering steps for a local projection operation of a single plane in 4D. ....... 100
Figure 5.9 Timings measured for different wavelet filters. ........................................ 104
Figure 5.10 Timing charts comparing transformation times. ..................................... 105
Figure 5.11 Comparison of timings for local reconstruction operations. ..................... 107
Figure 5.12 Comparison of timings for complete wavelet transformations. ............... 108
Figure 6.1 Cylinder-plane parameterization. ............................................................ 110
Figure 6.2 Foreshortening occurring with the cylinder-plane parameterization. ......... 111
Figure 6.3 Sphere-plane parameterization. .............................................................. 112
Figure 6.4 Schematic illustration of the data structure for in-core compression. ........... 117
Figure 6.5  Schematic illustration of the custom file format. .........................121
Figure 7.1  Kaidan Magellan 2500 motorized object rig. ............................126
Figure 7.2  Sony DFW-X700 camera with XGA resolution and FireWire interface. .127
Figure 7.3  Image series of the planar calibration object. .........................128
Figure 7.4  Comparison of positional parameters. ................................128
Figure 7.5  Difference measurements of the first experiment in Figure 7.4. ....129
Figure 7.6  Difference measurements of the second experiment in Figure 7.4. .129
Figure 7.7  Comparison of the calibration pixel errors. ..........................130
Figure 7.8  The Frog light field. ......................................................130
Figure 7.9  The Husky light field. ......................................................131
Figure 7.10 The Ganesha light field. ...............................................131
Figure 7.11 The Haathi light field. .....................................................131
Figure 7.12 Effect of the HSDI using a 5-tap Gaussian as interpolation filter. .....135
Figure 7.13 Effects of a single local update operation. ............................136
Figure 7.14 3D visualization of input camera poses and test locations. .........139
Figure 7.15 Visualizations of differences of signal energies (1st experiment). ......140
Figure 7.16 Visualizations of SNR measurements [dB] (1st experiment). ..........141
Figure 7.17 Visualizations of SNR measurements [dB] of all 10 reference sets. ....142
Figure 7.18 SNR gain at the update position (1st experiment). ....................143
Figure 7.19 Visualizations of differences of signal energies (2nd experiment). ......144
Figure 7.20 Two visualizations of SNR measurements [dB] (2nd experiment). ....144
Figure 7.21 Pre-update images (2nd experiment). ................................146
Figure 7.22 Post-update images (2nd experiment). ................................146
Figure 7.23 Example (1) of an accepted local update operation. ..................148
Figure 7.24 Example (2) of an accepted local update operation. ..................148
Figure 7.25 Example (3) of an accepted local update operation. ..................148
Figure 7.26 Example (4) of a rejected local update operation. ....................149
Figure 7.27 Example (5) of a rejected local update operation. ....................149
Figure 7.28 Example (6) of a rejected local update operation. ....................149
Figure 7.29 Regular evaluation grids used for the overall SNR measures. ........152
Figure 7.30 Evolution of the overall SNR. ........................................153
Figure 7.31 Evolution of the angular spacing between neighboring camera locations. 153
Figure 7.32 Impact of compression on visual quality (level-1, Haar). ..............156
Figure 7.33 Impact of compression on visual quality (level-1, Daubechies-4). ....157
Figure 7.34 Impact of compression on visual quality (maximal level, Haar). ....158
Figure 7.35 Impact of compression on visual quality (maximal level, Daubechies-4). 158
Figure 7.36 Comparison of SNR trends. ............................................159
Figure 7.37 Comparison of in-core compression factors. ..........................159
Figure 7.38 Comparison of different compression factors. ........................160
Figure 7.39 Comparison of memory consumption. ................................161


# Tables

| Table 1.1 | Example key data of extents of light fields. | 4 |
| Table 2.1 | Classification of image-based models via model characteristics. | 11 |
| Table 2.2 | Comparison of different IBR techniques. | 28 |
| Table 2.3 | Classification of IBMR methods via dimensionality of the representation. | 30 |
| Table 2.4 | Properties of discussed parameterizations. | 37 |
| Table 4.1 | Examples of orthogonal polynomials. | 54 |
| Table 4.2 | Parameter sets describing the Truck and the Falls light field. | 83 |
| Table 5.1 | Assembly of transformation costs of problem settings from 1D to 4D. | 103 |
| Table 6.1 | Compression performance in terms of computational costs. | 122 |
| Table 6.2 | Compression performance in terms of compression factors. | 123 |
| Table 7.1 | Parameter set describing the Frog and the Husky light field. | 132 |
| Table 7.2 | Parameter set describing the Ganesha and the Haathi light field. | 132 |
| Table 7.3 | Parameter set describing the light field used in Figures 7.12 and 7.13. | 137 |
| Table 7.4 | Parameter set describing the light field used in Section 7.4.2. | 138 |
| Table 7.5 | SNR gain [dB], measured at the update position (1st experiment). | 141 |
| Table 7.6 | SNR gain [dB], measured at the update position (2nd experiment). | 145 |
| Table 7.7 | Parameter set describing the light field used in Section 7.5. | 147 |
| Table 7.8 | Acceptance decisions based on RMSE values of examples (1)-(6). | 150 |
| Table 7.9 | Parameter set describing the light fields used in Section 7.6. | 151 |
| Table 7.10 | Parameter set describing the light fields used in Section 7.7. | 154 |
| Table B.1 | Parameter set describing the input images of the Truck scene. | 169 |
| Table B.2 | Parameter set describing the input images of the Falls scene. | 170 |
| Table B.3 | Parameter set describing the input images of the Frog scene. | 170 |
| Table B.4 | Parameter set describing the input images of the Husky scene. | 171 |
| Table B.5 | Parameter set describing the input images of the Ganesha scene. | 171 |
| Table B.6 | Parameter set describing the input images of the Haathi scene. | 172 |
CHAPTER 1

INTRODUCTION

Probably the essential objective target of computer graphics research is not just the generation of images on computers. It rather was—and, of course, still is—the quest for photorealism. It is not just the good looking, artificial image the whole community is working for, it is the image that looks as if it was real, shot using a traditional photo camera. First 2-dimensional computer graphics methods mark the commencement on the path towards artificial but photorealistic images. Many different techniques have been introduced along this path in order to make generated images look more authentic. Namely the results of shading and lighting techniques, e.g. global illumination methods like raytracing or radiosity, allowed for a major step forward towards photorealism.

Despite all these substantial improvements, it is still a very difficult task to achieve persuasive photorealism. Firstly, the traditional modeling and rendering methods require detailed and accurate models for the geometric proportions and for the lighting conditions. Thinking of fur and hair, human eyes or fire and explosions, creating these models may be very hard. Secondly, the solutions and their algorithmic representations available so far are still computationally costly, i.e. the process of image generation may be very time consuming.

Texture mapping and all its numerous enhancements, e.g. projective textures, environment mapping or bump mapping, have always been an additional, effective manner to add realism to generated images. Using real photographs clearly brings computer graphics closer to photorealism.

It is exactly this simple insight that forms the basic concept of all image-based modeling and rendering methods: A real image contains a huge amount of information about a scene’s appearance. A collection of real images contains even more, often highly redundant information and additionally enables to overcome the severe limitation of a fixed viewing direction. In fact, image-based modeling (IBM) and image-based rendering (IBR) have proved to be a very promising alternative to the traditional modeling and rendering methods, mainly due to the high degree of photorealism provided.
Light field rendering [75] and The Lumigraph [45] both presented in 1996 stand for two very similar core methods in the field of IBR, which have attracted a lot of attention since then. Much of the follow-up work of these two publications suggested various ways and tools to efficiently reconstruct approximations of the underlying plenoptic function—mostly from a set of dense scene samples and sometimes in combination with rough scene geometry. As a result, the initial concepts have been improved significantly in terms of reconstruction quality and storage efficiency, too. Chapter 2 gives an overview of this literature. In particular, Section 2.2 discusses Light field rendering and The Lumigraph in detail.

In spite of a few exceptions, however, relatively little work has been devoted to the development of a fast and easy-to-handle data acquisition scheme for light field rendering systems—even though there is a clear need for such a scheme. In the context of the Digital Michelangelo Project [74] Marc Levoy reports about the challenges one has to deal with when acquiring high-resolution light field data of a real statue. Besides all logistical and technical issues, time turned out to be the most important factor.

1.1 MOTIVATION & GOALS

All recent advances in the field of IBR notwithstanding, the literature still lacks ideas for simple yet fast light field (LF) data acquisition systems. A potential solution needs to allow for quite a number of requirements the most important of which are given below.

♦ Flexibility:
The image acquisition construction used for the large-scale light fields in the context of the Digital Michelangelo Project [74, 76]—a video camera mounted on a computer-controlled planar gantry—is certainly not the most flexible setup. It produced superb results at the expense of a heavy, huge and expensive design (Figure 1.1). A more flexible solution should be cheaper and a lot simpler to handle.

♦ Applicability:
Flexibility in terms of handling may also include the explicit wish for a solution applicable not only to synthetic scenes but to real-world scenes, too—and, preferable, by using a simple, consumer-style hand-held camera. Clearly, useful applicability is then only given in combination with fast data processing.

These requirements, listed from a user’s point of view, may be transformed into more technically oriented necessities.

1.1.1 Acquisition Scheme

An acquisition system ideally supports the user with some sort of user guidance during the acquisition process. Additionally, it tries to run as autonomous as possible. To this end, it may perform the following actions.

♦ Automatic image sample selection:
Starting from a small and incomplete set of input images of a 3-dimensional scene, sampled at irregular positions, the system continuously grabs new image samples. It decides automatically and progressively whether or not to accept a new image sample, depending on some analysis regarding the amount of new information to be learned. Accepted images are incrementally inserted into a suitable representation, rejected images are discarded.
1.1 MOTIVATION & GOALS

FIGURE 1.1 The Digital Michelangelo Project: Acquisition of a large-scale light field of Michelangelo's statue of Night [©Levoy].

a) Positioning of the gantry around the statue of Night.
b) Acquisition planning (7 light slabs).
c) Camera plane.
d) Camera motions.

- Visual feedback:
  Generally, the core data of a light field rendering system is represented in a 4-dimensional abstract space, as is reviewed in detail in Section 2.2. Obviously, it is very difficult to meaningfully visualize this multi-dimensional data set itself or any changes in it. Therefore, visual feedback of progress and changes throughout the acquisition process must be accomplished by other more intuitive means. A short discussion may be found in Section 3.4.

- Stop criterion:
  The termination of the acquisition process might be initiated automatically by the system shortly after a stop criterion has been met. Of course, such a termination criterion needs to be directly controllable by a user. From the user's point of view the indication of a possible acquisition termination only, i.e. a notification mechanism, might be preferable.

1.1.2 Suitable Representation

The internal representation of the core data probably is the crucial issue to be addressed. Firstly, as is mentioned above, the problem of multi-dimensionality is an integral characteristic in the field of IBR in general, particularly for LF rendering systems. Secondly, a representation for an acquisition system has to support a highly dynamic setting. It needs to allow for update operations at any time and, for the purpose of rendering, it needs to allow for fast data read-out, too. Additionally, with respect to a continuous data refinement, all update operations are advantageously performed in an incremental manner.

Besides the high dimensionality and all the above mentioned operational issues, there is an additional, very important structural issue to be addressed. Compressed core data must not invalidate nor override any of the operational features named above, i.e. behavior and functions of the internal representation must stay independent of the data's structure. More technically speaking, the representation's interface must not change when compressing or de-compressing the core data.
1.1.3 Control of Memory Consumption

As is already brought up in the previous paragraph, the problem of compression is an important matter to be involved. Moreover, in the field of IBR, in-core compression has proved to be of critical importance for rendering especially when no geometrical scene information is involved at all as is the case for LF rendering. Table 1.1 below lists examples of the total number of input images and the resulting—partially immense—data set sizes of light fields originally presented in [75].

However, memory control mechanisms for IBR systems may conceptually be split up into the following two groupings.

- **Active memory control:**
  Data set compression of any kind may be classified as a means for active control of memory consumption. The challenge thereby is not to compress LF core data for permanent storage. It rather is the control of the memory consumption during the acquisition procedure and during reconstruction, i.e. rendering. This is referred to as the *in-core compression*. Additionally, a compression scheme must not only be efficient with respect to memory usage but with respect to data access time, too.

- **Passive memory control:**
  Besides an effective compression scheme one might think of additional data reduction mechanisms. An intelligent, automatic image selection scheme, as is already addressed in Section 1.1.1, helps to only process useful and important input data and to discard all other data, thus reducing the amount of data to be stored in some sense.

The problem of active memory control, i.e. compression, is discussed in detail in Section 6.2. A notion of a passive memory control scheme is presented in Section 3.3 and deepened in Section 4.8.

<table>
<thead>
<tr>
<th>Category</th>
<th>Buddha LF</th>
<th>Kidney LF</th>
<th>Hallway LF</th>
<th>Lion LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total input images</td>
<td>256</td>
<td>4096</td>
<td>8192</td>
<td>2048</td>
</tr>
<tr>
<td>Raw data set size [Mb]</td>
<td>50</td>
<td>201</td>
<td>1608</td>
<td>402</td>
</tr>
</tbody>
</table>

1.2 CONTRIBUTION

In correspondence with the motivation given in the previous section, the contributions of this thesis can be summarized as follows.

- **Unified, hierarchical data representation:**
  A so-called extended multiresolution analysis (eMRA) is proposed as a hierarchical representation for the core data of a LF rendering system. It combines a conventional wavelet transform with a hierarchical scattered data interpolation method. In theory, it is applicable to scenarios of any dimensionality and is therefore well tailored to the 4-dimensional space of LF rendering systems. The usage of both a specific wavelet filter and a specific interpolation filter is not predetermined. In fact, the presented eMRA accepts any orthogonal wavelet and any reasonable interpolation filter. After
all, it serves as a unified representation for sampling, interpolation, dynamic updates, rendering and compression.

- **Fast local reconstruction operator for rendering:**
The employed representation, an extended 4-dimensional wavelet transform, allows to design highly efficient local reconstruction operators for rendering. Generally speaking, rendering means reconstructing one or more 2-dimensional views out of the 4-dimensional, wavelet-transformed data set. Clearly, complete back-transforms of the core data must be avoided. The local operator computes image reconstructions locally in the wavelet domain by exploiting the spatial localization of wavelets. Therefore, it does not require a global inverse transform, which greatly reduces the computational complexity.

- **Fast local projection operator for incremental data insertion:**
A local operator performing a decomposition locally in the wavelet domain can be designed in the exact same manner. This local projection operator is needed for incremental updates in the core representation. Such an update requires the insertion of data in 2D space, i.e. an image or more precisely a difference image, into the 4-dimensional, wavelet-transformed data set. Again, complete back-transforms in order to perform an incremental update with a subsequent re-transform of the core data must be avoided. The local projection operator computes a local decomposition of the input data directly in the wavelet domain and weaves it into the 4-dimensional representation, thus eminently reducing the computational costs.

- **Dynamic acquisition scheme:**
The design of the eMRA together with the set of local operators make a highly dynamic acquisition system possible. The first processing stage covers the calculation of the eMRA by combining scattered data interpolation and wavelet transform. This first step is computed based on an initially small set of input images. During the second stage the initial representation is continuously and selectively refined through progressive image data acquisition. This refinement highly relies upon the locally working operators.

- **Optimized compression scheme:**
Employing an extended wavelet transform as representation for the core data, the framework for a well-fitting, lossy in-core compression by means of coefficient thresholding is directly given. A suitable data structure enables both local operators to also work on the set of thresholded coefficients, i.e. the compressed representation. Permanent storage of the representation on disk requires a reorganization, i.e. a serialization of stored coefficients, for the purpose of progressive read-back. Once written to disk, and additional lossless compression utilizing a Lempel-Ziv coder, for instance, may be preformed.

- **Experimental validation:**
All of the above mentioned components are integrated in a prototype system which is extensively evaluated. This system is built to potentially use input imagery from two sources, namely (1) synthetic, computer-generated images as well as (2) real-world image data. Both alternatives are heavily used to test the system and its individual components. Imagery of the latter source, however, is recorded and processed in an experimental setup in order to validate the system in a real-world environment.
The proposed hierarchical representation for light field data in combination with the set of local decomposition and reconstruction operators form the core contribution of this thesis. They form a powerful framework for interactive light field data acquisition and its processing.

However, it is important to note that this thesis does not cover issues related to the exact position and orientation tracking of a hand-held camera used as input device. Consequently, there is neither a discussion nor an evaluation of available and potentially suitable tracking systems included.

1.3 OUTLINE & ORGANIZATION

The remainder of this thesis is organized as follows:

- Chapter 2, *Image-based Methods*, gives a survey of the field of image-based methods in computer graphics used in the context of this thesis. After a short, introductory overview of IBM as a whole, the two fundamental image-based rendering methods in the connexion of the thesis are revisited: Light field rendering [75] and the Lumigraph [45]—their similarities as well as their differences. This comparison is followed by a state-of-the-art discussion of a collection of specific IBMR methods, divided into four groups, namely acquisition, parameterization, compression and wavelets.

- Chapter 3, *System Overview*, gives an overview of the entire system as a whole and introduces all its conceptual components, with an emphasis on the system's core and its two modes.

- Chapter 4, *Data Representation*, first summarizes the important results and properties of the wavelet theory as well as the approximation theory that are needed later on in this chapter. It then reviews hierarchical interpolation schemes and hierarchical representations in the field of IBM before discussing the construction of the so-called eMRA. The functioning of the incremental updates on top of the eMRA in combination with the oracle mechanism and a presentation of the visual performance of all of this operators conclude this chapter.

- Chapter 5, *Local Operators*, introduces the concept of local operators in the context of wavelet transforms. It discusses both the local reconstruction operator needed for rendering as well as the local projection operator used for performing incremental updates. This chapter concludes with an analysis of the computational performance of the local operators.

- Chapter 6, *Implementation Details*, discusses in detail two additional, technically important modules: Firstly, the employed parameterization spaces with their pros and cons, and secondly, the data compression and all its relevant aspects, namely in-core data structure, file structure and compression performance.

- Chapter 7, *Evaluation*, presents and discusses the results obtained when applying the techniques and methods introduced in the previous chapters to real-world video images. It first gives an overview of the tools and techniques used to acquire and calibrate the real-world image material.

- Chapter 8, *Conclusions & Future Work*, concludes the thesis with a summary and the presentation of findings, and suggests directions for future work.
The Appendix chapters include a commented version of one of the core routines, a compilation of the parameter sets of all lots of available input imagery, nomenclature, symbols and abbreviations used in preceding chapters as well as all references.
Seite Leer / Blank leaf
The traditional procedure of image generation in computer graphics is often referred to as the geometry-based approach. It is a two-stage procedure, the first of which covers the compilation and suitable conditioning of input data into the representation used for the second stage. The result of this first stage, the virtual model of a 3D scene, is mainly made up of a polygon-based description of the scene geometry, but does also comprise the characterization of additional, important attributes that might be summed up as an object's appearance, e.g. reflectance and surface properties, lighting conditions, etc. The second stage covers the rendering, i.e. the actual image generation, based on the model representing a 3-dimensional scene, employing the traditional rendering pipeline.

On the one hand, the image-based modeling and rendering approach does also follow this two-stage procedure. On the other hand, IBMR fundamentally differs from the traditional approach in that both geometry and appearance of a scene and its objects are derived from photographs. Since a relatively large amount of imagery—photographic, but also synthetic—is being used, IBMR techniques achieve an unprecedented degree of photorealism. Moreover, they often allow for shorter modeling times and offer efficient rendering speeds. As an example, Figure 2.1 shows a set of image-based renderings of different models that may nowadays be calculated at interactive frame rates.

Input images for the image-based modeling stage may be used to determine a broad variety of scene attributes, namely the scene appearance, scene geometry, if needed at all, lighting and reflectance characteristics, and also kinematic properties. Appearance of a scene in available views, i.e. appearance represented by means of the model, is then used to determine appearance in novel views during the rendering stage.

As a matter of course, a full variety of different ways of combining input images into models and subsequently rendering novel views out of these models has emerged during the past few years of research in the field of IBMR. In essence, it is the idea of taking at least two images and rendering novel views through smart combinations of image warping and blending that is common to all of them.
The question might arise how to classify all these different image-based methods in order to get an overview. There is no unique but several plausible ways to do so. A first categorization would take the trade-off between the amount of geometry-based input compared to the amount of image-based input into account. This classification is also referred to as the rendering spectrum. A second categorization might be based on the underlying image-based models and what they allow for, e.g., regarding viewpoint motions. Lastly, IBMR methods might also be grouped according to the processing of input images, e.g., direct manipulation or reconstruction of higher-level representations.

The following paragraphs briefly discuss these three categorizations in order to firstly, file the overview given in the succeeding Section 2.1 within the vast field of image-based methods, and secondly, in order to precisely delimit the scope of this thesis. Although there exists a close connection between IBMR and global illumination methods, note that the field of image-based lighting techniques is not included in the following classification. Image-based lighting deals with the combination of real and synthetic graphical models with consistent illumination. Consequently, the overview given in Section 2.1 does not take image-based lighting methods into account.

**Classification through rendering spectrum.** The left extremum of the rendering spectrum represents pure geometry-based methods, while the opposite right end represents pure image-based methods, as is shown in Figure 2.2. A spot somewhere in between depicts a hybrid method that uses concepts of both approaches. A geometry-based method moves toward image-based methods by adding discrete, raster-based information, e.g., texture images, and vice versa, an image-based method dislocates towards geometry-based methods by adding geometrical information.

To single out a few examples, Light field rendering [75] stands for an image-based method that uses absolutely no information about the geometrical appearance of a 3D scene—only positional and orientational information of the camera locations of all input images are needed. Contrary to that, a method using the original texture mapping technique [19] relies on a detailed geometrical scene description and uses very few raster-based information as texture maps. The Surfels publication [99] as well as the Layered depth images [111] may be considered representatives of hybrid methods using a somewhat balanced amount of both geometry and imagery as input.
Classification through model characteristics. Not every model allows for the same interaction. Dissimilarities may be found in a variety of features. Table 2.1 restricts its short overview on the three characteristics 1) view point movement, 2) geometry involved and 3) scene lighting. Model a) in Table 2.1 represents a purely geometry–based model, i.e. it would need to be positioned on the far left side in the above figure. Of course, such a model allows for free view point positioning due to detailed and complete geometry knowledge. Model b) combines geometry and images, e.g. via texture mapping, thereby enhancing realism, but at the expense of fixed lighting conditions. Model c) still allows for continuous view point movements and uses only local geometry information. The Layered depth images [111] may be classified in this group. Light field models (d) do not rely on any geometric information while still allowing continuous view point movements. Contrary to that, Movie–Maps [81] as well as panoramic models much like the Quicktime VR system [22] are restricted to limited movements. Finally, a single image, the model g), is not a very flexible representation at all.

**TABLE 2.1** Classification of image–based models via model characteristics.

<table>
<thead>
<tr>
<th>Model</th>
<th>(1) movements</th>
<th>(2) geometry</th>
<th>(3) lighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Geometry &amp; materials</td>
<td>continuous</td>
<td>global</td>
<td>dynamic</td>
</tr>
<tr>
<td>b) Geometry &amp; images</td>
<td>continuous</td>
<td>global</td>
<td>fixed</td>
</tr>
<tr>
<td>c) Images &amp; depth</td>
<td>continuous</td>
<td>local</td>
<td>fixed</td>
</tr>
<tr>
<td>d) Light field</td>
<td>continuous</td>
<td>none</td>
<td>fixed</td>
</tr>
<tr>
<td>e) Movie map</td>
<td>discrete</td>
<td>none</td>
<td>fixed</td>
</tr>
<tr>
<td>f) Panorama</td>
<td>rotation</td>
<td>none</td>
<td>fixed</td>
</tr>
<tr>
<td>g) Image</td>
<td>none</td>
<td>none</td>
<td>fixed</td>
</tr>
</tbody>
</table>
Note that the usage of input imagery with given, fixed lighting automatically determines lighting conditions in renderings from such a model—as long as no relighting techniques are being applied. This is reflected in the last column of Table 2.1.

Furthermore, notice that Table 2.1 shows a comparison of a few models from an IBMR point of view. Although this is a meaningful compilation, the models a) and b) might rather be referred to as geometry-based models. Likewise, model c) might be denoted a hybrid model.

**Classification through input processing.** Some of the approaches in the field of IBMR directly manipulate single or multiple input images in order to generate novel views. Others build approximate reconstructions of higher-level delineations, namely the Plenoptic function [2] which is a 5-dimensional function representing the intensity of light observed from every position and every direction in 3-space. The Plenoptic function will be reviewed in Section 2.1.

Figure 2.3 gives an overview of the grouping of IBMR methods via diverse input image processing. For the sake of completeness, the primary partitioning into techniques generating novel illumination and techniques generating novel views has been included. As is stated before, the field of image-based lighting techniques will not be further dealt with in the scope of this thesis.

The two most prominent representatives of IBMR techniques generating novel views by approximating the Plenoptic function are unquestionably Light field rendering [75] and the Lumigraph [45] which will both be reviewed in Section 2.2. Methods of the second branch generate novel views by resampling one or more input images using appropriate warping functions, thus directly manipulating the input images. Some use single input images, e.g. environment maps as first presented in [9], or a collection of input images that can be mapped on a sole image plane. The latter holds for image sequences formed by a rotating camera around its optical centre, as it is the case for panoramas, for instance. Quicktime VR [22] is probably the most prominent example for a system mapping a set of input images on a single image plane and recovering views by applying an inverse mapping.

Methods manipulating two or more input images shot from different camera locations apply image understanding techniques in order to either interpolate or reconstruct novel views. If the correspondences between image pixels of adjacent viewpoints—which will commonly have a very similar appearance—are known, intermediate views may be approximated through pixel interpolation. A well-known example for image interpolation techniques has doubtlessly been presented in [23]. Image reconstruction techniques are based on the observations that accurately defined correspondences exist between the positions of pixels on the image plane representing the same points in 3D space sighted from different points of view. Given two views, any third view can be reconstructed, provided that the correspondences between image pixels as well as the camera’s internal and external parameters are known. This is the conclusion the authors came to who asked the question “What can two images tell us about a third one?” [39].

After having briefly discussed three possible categorizations of IBMR methods it is important to precisely delimit this thesis' scope. Since it introduces a dynamic light field representation for acquisition and processing, it is relatively easy to file this work, following the given categorizations. All systems closely related to the idea originally presented in the
2.1 OVERVIEW

Apart from all texture mapping techniques following the initial work of Blinn and Newell [9]—a comprehensive overview of texture mapping may be found in [52]—that may all be understood as early image-based approaches to computer graphics, there exists a publication which might be regarded as the first interactive, image-based rendering system very well worth mentioning. The Movie-Maps [81] presented in 1980 is a system that allows to interactively explore a large 3D space with photorealistic image quality. The input to the system was video material filmed by a camera mounted on a vehicle, driving down all the streets of downtown Aspen, USA. The video was then transferred to a video—
disc, allowing for random access, and was further enhanced with other information. A clever user interface made the virtual exploration of downtown Aspen possible, in real-time, with photorealistic image quality.

The same technology was adopted by Naimark to produce a moviemap of the San Francisco Bay Area from the air [96]. A special gyro-stabilized helicopter camera in combination with satellite navigation was used to film along a 2D grid pattern over the Bay Area, with the camera kept pointed toward the Golden Gate Bridge. A trackball interface again allowed for a real-time fly over of San Francisco. Figure 2.4 gives an impression of the system and the foregoing data capturing.

![FIGURE 2.4](image)

**FIGURE 2.4** Golden Gate fly-over, a moviemap of the San Francisco Bay Area [©Naimark].

**a)** Helicopter with gyro-stabilized motion picture camera system.
**b)** The flight plan, a 10 x 10 miles 2D grid, centered on the Golden Gate Bridge.
**c)** System's user interface with trackball controller.
**d)** The system on exhibit at the Exploratorium, San Francisco.

The new approach to *Near-field Photometry* [5] by Ashdown does not come up with a system as spectacular as these two moviemap applications. But, nonetheless, it is a very visionary work that anticipated many of the results presented synchronously years later in the Light field rendering and the Lumigraph publications. It therefore stands for an important milestone towards LF-like techniques.

Ashdown presents a new approach to near-field photometry and describes a novel near-field goniophotometer and an illuminance calculation method. From an IBMR point of view it is the model for a luminaire's near-field environment that makes up the interesting part of this work. The model is called the Helios approach and follows the helios concept presented in [93] where helios at a given point and in a given direction is defined as the factor $\pi$ times the luminous flux density in that direction per unit solid angle. Furthermore, the authors of [93] add that helios is not a characteristic of a surface but it is a condition at a point in the light field. In other words, luminance is an intrinsic property of the field of light surrounding an observer in 3-space. It is not a property of any surface, whether real or imaginary.

Based on the so-called Helios approach, numerous insights and side aspects are discussed some of which are listed below. Note that all of them are also fundamental to LF rendering systems.

- **Independence of geometry:**
  The helios approach involves measuring the field of luminous flux rather than the surface luminance or any other intrinsic property of the luminaire. Hence, no knowledge about the geometry of the luminaire nor of objects in the environment is required.
  The independence of scene geometry is a basic property and one of the major advantages not only of LF-like systems but of many image-based methods in general.
Measurement of the photic field:
Since the luminance along a ray is invariant in a non-participating medium such as a vacuum or air—over short distances—, the spatial distribution of the luminous flux surrounding a luminaire may be considered a 3-dimensional vector field of light, which is conceptually analogous to the 5-dimensional scalar irradiance field called Photic field [94] introduced in 1981. The measurement of the luminance along a ray may be done at any convenient point along the ray, e.g. at the intersection point with a closed convex surface surrounding a luminaire, with the restriction that the measuring point must be located outside an imaginary bounding volume fully enclosing the luminaire.

The geometrical setting such as planes used to parameterize rays in LF or Lumigraph like systems are equivalent to a cutout of such a surface. Additionally, the positioning of these parameterization bodies take advantage of the luminance invariance property.

Sampling and interpolation:
The helios approach requires an instrument capable of quickly and accurately measuring the luminance of geometric rays. Thereupon, the finite subset of measuring rays of the infinite number of rays must be determined that is sufficient to properly characterize the photic field surrounding a luminaire. The solution is based on a CCD photosensor, mounted on a rigid arm rotating in the vertical plane about the luminaire. After measurement, the illuminance calculation for a point on the surface surrounding a luminaire involves ray intersection computations. Such estimated rays do most likely not coincide with measured rays which, in turn, makes the determination of closest measured rays necessary, with subsequent bilinear interpolation.

The choice of a suitable ray parameterization, its grid spacing, or, in other words, the sampling distances on the parameterization surface with respect to input and output resolutions, together with suitable interpolation strategies for rays not hitting sample locations are foundational issues to all LF like systems.

Data compression:
The storage of a complete set of video frames measuring a luminaire’s near-field consumes a lot of memory. Hence, lossless as well as lossy image and video compression techniques are employed in order to eliminate both intraframe (spacial) and interframe (temporal) redundancy. Intraframe compression also includes downsampling, i.e. block-based averaging, thereby improving the signal-to-noise ratio of a frame.

Purely image-based representations produce large data sets. As a consequence, efficient data compression is crucial for their success. Many different compression schemes have been proposed, including various video compression techniques. Some of them will be reviewed in Section 2.3.3.

Implications for computer graphics:
As a comment on the helios approach it is stated that for any point in space the measured knowledge of the photic field can be translated into a photometrically accurate view of the environment as seen from that very point.

In essence, this is exactly the basic idea of many IBR techniques, namely sampling a field of light of some 3D scene via image acquisition and generating novel views using this acquired data.

Despite all these visionary thoughts regarding computer graphics they might rather be comprehended as conceptual spadework. Ashdown brings up possible embeddings of his
work in ray-tracing or radiosity techniques, but no precise idea of an image-based application.

However, Ashdown's notion of light in space follows the previously mentioned Photic field [94], a scalar irradiance field. The idea of light fields—as well as the phrase itself—, however, dates back to a paper by Gershun on radiometric properties of light in space [43] published in 1939, where light in space is understood as vector irradiance field. A third perception of fields of light gives the Plenoptic function [2] introduced in 1991. The Plenoptic function is the underlying idea of a whole class of IBMR methods, as has been shown in Figure 2.3. The goal of all of these rendering methods is to reconstruct a continuous representation of the Plenoptic function, given only a set of discrete samples.

The Plenoptic function as brought up in [2] is a 7-dimensional function \( P_{I7} \) (Equation 2.1) that describes a complete holographic representation of the visual world. It records the intensity distribution \( P \) of incident light rays at a viewing position \((V_x, V_y, V_z)\) in 3-space, that is parameterized using the spherical coordinates \( \theta, \phi \). The parameter \( \lambda \) takes the wavelength of impinging light into account, whereas the parameter \( t \) includes the dimension of time. Hence, the complete knowledge of the function \( P_{I7} \) would allow reconstructions of every possible view, at every moment in time, as seen from every position, at every wavelength. Such a complete representation would implicitly include a description of every possible photograph that could be taken at a particular moment in time—or, in the terminology of computer graphics, it describes the set of all possible environment maps for some 3D scene.

\[
P_{I7} = P(V_x, V_y, V_z, \theta, \phi, \lambda, t)
\]  

By only considering a snapshot of the function, the 7th dimension time may be eliminated. As a consequence, \( P_{I7} \) then describes a complete visual representation of a static world. An additional limitation to the three widely used color channels R, G and B instead of the full color spectrum further reduces \( P_{I7} \) to a 5-dimensional function. Figure 2.5 gives an illustration of the remaining five parameters. This reduction in dimensionality is done by all IBMR methods that try to reconstruct the Plenoptic function. Besides simplicity, this is due to limitations in acquisition and storage of a 7D function.

\( (V_x, V_y, V_z) \): viewing position \( V \) in 3D space
\( (\phi, \theta) \): direction of ray in spherical coordinates

**FIGURE 2.5** Parameters of the Plenoptic function, reduced to 5 dimensions.
2.2 LIGHT FIELD & LUMIGRAPH RENDERING

As has been mentioned before, many IBMR methods hark back to the Plenoptic function as their fundamental representation. So does *Plenoptic Modeling* [89], an image-based rendering system by McMillan and Bishop that has been presented shortly before Light field rendering and the Lumigraph. Besides their own system, the paper presents a consistent framework for the evaluation of image-based rendering systems, and gives a concise problem definition. The framework is based on the definition of the Plenoptic function and it is claimed that all approaches to IBR can be cast as attempts to reconstruct the Plenoptic function from a set of samples of it. Hence, the problem definition for IBR may be stated as follows: Given a set of discrete samples from the Plenoptic function, the goal is to generate a continuous representation of that function—as has been mentioned before. As a consequence, the three key steps of all IBR methods are 1) sampling, 2) reconstruction and 3) re-sampling of the Plenoptic function.

The system of McMillan and Bishop emanates from a series of reference images that represent a description of a real 3D scene. These input images are subsequently warped and combined in order to form representations of the scene from arbitrary viewpoints. The warping function is defined by image flow field information that is derived from the reference images. Alternatively, it can be supplied as an additional input.

A cylindrical projection is used as representation for plenoptic samples. These samples are acquired employing a video camera that is panned through a full circle. All necessary parameters are automatically estimated from the set of input images.

---

Discussing a complete list of IBR techniques, even in the field of LF like system, goes beyond the ambit of this Chapter 2. Surveys may easily be found. As an example [66] gives another, alternative classification for IBR techniques. Four basic categories, that are 1) non-physically based image mapping, 2) mosaicking, 3) interpolation from dense samples and 4) geometrically-valid pixel reprojection, are being identified and commented, with an emphasis on the fourth class. Clearly, Light field rendering and the Lumigraph are representatives of the third group.

The following Section 2.2 concentrates on Light field rendering and the Lumigraph, i.e. mainly on their similarities and differences. Significant follow-up work will be discussed in Section 2.2.4.

2.2 LIGHT FIELD & LUMIGRAPH RENDERING

Both publications, Light field rendering [75] and the Lumigraph [45] have remarkable similarities—even notations are similar. On the other hand, both papers make their unique contributions. Rather than explaining the two approaches one after the other in detail, this section first gives a brief description of the basic concepts and then contrasts these analogies with the differences.

2.2.1 Characterization

*Light field rendering* describes a robust method for generating new views from basically arbitrary camera positions without any depth information or feature matching, simply by combining and resampling a set of available input images.
This technique is based on a high-dimensional function, the so-called light field, that completely describes the flow of light through unobstructed space in a static scene with fixed illumination. All input images are being interpreted as 2D slices of this 4D function.

As a matter of principle, the Lumigraph does also not rely on geometric representations in order to capture a synthetic or real-world scene’s complete appearance and render new images of it thereafter. Moreover, it also samples and reconstructs a 4D function, called lumigraph, that describes the flow of light through space.

Nonetheless, it is important to note that there are considerable differences the most substantial of which affects the usage of geometrical information about an object or a scene. While the Light field rendering system does not use any geometry at all, the Lumigraph does. It computes a crude approximation of the 3D shape only from the available input imagery. This 3D information is mainly used for the depth correction procedure, as will be discussed in Section 2.2.3.

2.2.2 Similarities

Clearly, both approaches benefit from all advantages that are common to IBR methods in general. The most important and well-known advantages are:

- **Simpler computation**, as compared to the traditional rendering pipeline.
- **Computational cost independent of scene complexity**.
- **Computational cost independent of scene appearance**, i.e. material properties, etc.

As a matter of course, Light field rendering and the Lumigraph also share the same disadvantages and limitations which are again common to most LF-like IBR techniques.

- **Limitation to static scenes**: This restriction is the direct consequence of omitting the last parameter \( t \) in the Plenoptic function (Equation 2.1). Of course, the reduction in dimensionality by fixing time comes at the expense of not being able to handle dynamic scenes.

- **Fixed lighting conditions**: The limitation to fixed lighting conditions may be understood as a consequence of only considering a snapshot in time of the Plenoptic function, too, namely in case of dynamically changing lighting conditions, e.g. moving light sources, changing lighting colors over time, etc. However, there are ways to relax this restriction. For instance, an image-based rendering system with controllable illumination has been presented in [136]. This system allows the illumination to be changed interactively by the user. Illumination calculations are done without recovering or using any geometrical information, e.g. depth or surface normals, and the resulting images are physically correct. This is being achieved through measuring of the BRDF [65] of a scene as seen through each pixel window of input images. Instead of directly storing the raw BRDF tables—which would require enormous additional storage space—, they are transformed to the spherical harmonics domain [32] in order to only store a few coefficients of the spherical harmonics per pixel.

- **High storage costs**: The sampling and subsequent storage of the Plenoptic function necessitates a considerable amount of storage space, due to its high dimensionality. In fact, there exists a well-known trade-off between memory costs, i.e. the number of input images used,
and the amount of geometrical knowledge introduced. Pure image-based techniques such as Light field rendering rely on no geometrical information at all, thus demanding high storage costs. Hybrid IBR techniques try to decrease the amount of memory needed by building a geometrical model from the input images.

Besides advantages and disadvantages common to many LF like IBR methods, Light field rendering and the Lumigraph share more similarities the most important ones are being discussed below.

**Processing & rendering pipeline.** Both publications describe a complete working system, from capturing of input samples to generating new views. Moreover, not only the rendering stages resemble each other a lot. In fact, both systems follow almost the same steps throughout the whole processing pipeline, that are:

1) **Capturing** of input images, in order to generate new views of a scene from arbitrary camera positions, without extracting depth or other geometrical information.

2) Employment of a suitable parameterization for adequately describing the rays of light measured by the captured images.

3) Construction of an internal representation that describes the flow of light through unobstructed space in a static scene with fixed lighting. Both representations describe a subset of the Plenoptic function and are actually equivalent—one is called a Light field, the other a Lumigraph.

4) **Rendering** of new images from these representations by selecting the subset of affected rays of light and appropriately interpolating amongst them, which genuinely means re-sampling of the function serving as representation.

**Parameterization.** Since both Light field rendering and the Lumigraph try to reconstruct portions of the Plenoptic function for static scenes with fixed lighting conditions, they both need to deal with the 5-dimensional function representing the radiance at a point in 3-space in a certain direction (Figure 2.5). However, the 5D representation may be reduced to 4D using the luminance invariance property that has been discussed earlier in the connexion of Ashdown’s work on Near-field Photometry (Section 2.1). By only considering the light leaving the convex hull of a bounded object and taking advantage of the luminance invariance property in regions free of occluders, all rays of light, i.e. lines in space, may be parameterized by means of intersections on any surface surrounding the objects of interest. Both publications use the setting shown in Figure 2.6, which is called a light slab in [75]. The intersection points of a ray on two parallel planes are used to parameterize the ray, which yields two parameters $s$ and $t$ on the front plane ($st$-plane) and two parameters $u$ and $v$ on the back plane ($uv$-plane), therewith describing the space of rays. Intuitively speaking, a light slab represents the beam of light entering a quadrilateral and exiting another quadrilateral. The objects of interest are usually located near the back plane.

Naturally, many other settings for the parameterization of lines in 3-space would be feasible, too. The 2-plane parameterization was mainly chosen because of its computational simplicity. The inverse transformation from image coordinates $(x, y)$ to $(s, t, u, v)$, for instance, can be done using texture hardware. Alternative settings using cylindrical or spherical coordinates, i.e. a surrounding sphere, require substantially more computation, since angle calculations for ray intersections will be involved. Section 2.3.2 will discuss a few alternative parameterizations that have been proposed.
L(s, t, u, v) \text{ : ray of light, parameterized by } s, t, u, v

**FIGURE 2.6** The light slab, a 2-plane Light field parameterization to describe the space of rays.

Note that a single light slab only covers a very limited viewing area. In order to parameterize all possible views of an object, it takes multiple light slabs, i.e. six for full coverage, four light slabs for the four sides of a cube surrounding the object and additional two to take care of the top and the bottom face of the cube. A spherical setting, per contra, would inherently offer full coverage, without any discontinuous junctures.

The parameterizations employed in the context of this thesis will be given in Section 3.2 of the overview chapter. Furthermore, Section 6.1 in the chapter reviewing implementation details discusses them in greater detail.

### 2.2.3 Differences

Besides all similarities, Light field rendering and the Lumigraph each make their own contributions. Accordingly, there are important dissimilarities to note which will be commented on in the following.

**Acquisition procedure.** While both systems exploit real imagery as input material—in fact, IBMR methods most effectively benefit from their advantages when using real photographs—, the two procedures for image acquisition greatly differ. Since Light field rendering uses a computer-controlled camera gantry whereas the Lumigraph system uses a hand-held video camera, this comes at no surprise. Figure 2.7 gives an impression of the two different setups for image acquisition.

Note that an alternative method to create a Light field or a Lumigraph would be to choose a sampling pattern in 4D space and find the radiance value for each line sample as is stated in [75]. For virtual environments, this may be done easily using a ray-tracer. In a real environment, it could be done using a spot radiometer. However, the far more practicable way is to gather a collection of images.

Assembling a large number of images of a 3-dimensional scene requires to take care of quite a number of smaller problems, such as including fixed yet proper lighting, avoiding shadows imposed by moving parts of the acquisition setup, accounting for intrinsic parameters of the real optical system, and most importantly, measuring the camera’s pose, i.e. its position and orientation, in an accurate way. As has been mentioned before, the two systems follow quite different approaches to tackle these problems.

The Light field rendering system utilizes a computer-controlled camera gantry, based on a modified motion platform with additional stepping motors (Figure 2.7, a). This setup allows to digitize an object on a regular grid with known camera pose.
The Lumigraph system follows a rather inexpensive approach by moving a hand-held video camera through the scene. Consequently, the camera’s pose needs to be estimated for each frame. To this purpose calibration markers in a specially designed data capturing stage are necessary (Figure 2.7, b). An additional challenging problem involved is the interpolation of the 4-dimensional Lumigraph from scattered data because of the unstructured input.

**Re-binning & scattered data interpolation (Lumigraph).** The system described in the Lumigraph paper applies a pre-processing step right after image acquisition that is called re-binning. This procedure is about re-sampling the unstructured input images. In other words, input images are resampled to the 2-plane parameterization. Since the Light field rendering system places the camera on a regular grid while digitizing the object, this regularization procedure is not necessary.

The Lumigraph paper precisely describes the re-binning step as an approximation theory problem. In a discrete computational environment, an initially unknown and continuous 4-dimensional Light field or Lumigraph function $L$ can be approximated as a linear combination of suitable basis functions $B$ and their associated coefficients $x$. To this purpose, 4-dimensional basis functions that are centered at grid points of the 4D regular grid are reasonable in the present context. The continuous, projected version of $L$, denoted $L_{\text{proj}}$, can then be expressed as a linear sum, as is shown below:

$$L_{\text{proj}}(s, t, u, v) = \sum_{i=0}^{M} \sum_{j=0}^{M} \sum_{p=0}^{N} \sum_{q=0}^{N} x_{i, j, p, q} \cdot B_{i, j, p, q}(s, t, u, v)$$

In this case, the grid resolution in $s$ and $t$ dimensions is chosen to be $M$, whereas grid resolution in $u$ and $v$ is fixed to $N$. The optimal coefficients $x$, that are in fact multi-channel data values in R, G and B, are given by the inner products of the function $L$ with the duals of the basis functions, if the $L^2$-norm is chosen as distance metric:

$$\hat{x}_{i, j, p, q} = \langle L, \hat{B}_{i, j, p, q} \rangle$$

$$= \int L(s, t, u, v) \hat{B}_{i, j, p, q}(s, t, u, v) dsdtdudv$$
Since the continuous form of \( L \) is unknown, the integral in the previous Equation 2.3 must be evaluated with only a finite number of discrete samples of it. This evaluation using the set of available discrete samples only will be commented on in Section 4.5, Equations 4.40 in particular. Note that every pixel in an input image represents such a sample \( L(s_k, t_k, u_k, v_k) \).

Due to the unstructured input, available samples are generally not evenly spaced nor are they evenly distributed. As a consequence, a multi-dimensional scattered data interpolation is necessary, mainly in order to fill in undefined regions where no measured samples are accessible. The solution proposed is a splat–pull–push technique that includes the following three stages:

1) **Splat step:**
   Estimation of the coefficients \( \tilde{\xi}_{i,j,p,q} \) by approximately computing the integral in Equation 2.3 using Monte–Carlo integration (Equation 4.40, Section 4.5).

2) **Pull step:**
   Calculation of a whole pyramid of lower resolution approximations of the function \( L \) in order to make the gaps of missing data smaller and smaller.

3) **Push step:**
   Successive blending of higher resolution approximations with lower resolution approximations in order to fill regions of missing data in the higher resolution version, while leaving already known data untouched.

The wide field of approximation theory will be briefly discussed in Chapter 4. In particular, scattered data interpolation techniques suitable for multi-dimensional problems, including the splat–pull–push scheme, will be reviewed in Section 4.5 whereas the construction of the representation developed in the course of this thesis—which is based on these schemes—will be introduced in Section 4.6.

Note that this regularization procedure done in the Lumigraph system potentially degrades the overall quality of the representation, since it appends an additional reconstruction and sampling step as has been noticed in [12].

**Depth–correction (Lumigraph).** During construction of the Lumigraph or Light field function as well as throughout rendering from this function, calculated rays in space will most likely not precisely hit grid points of the chosen line parameterization as is shown in Figure 2.8 a). Thus, interpolation, that is, filtering with some suitable kernel, is necessary. Adopting a simple box basis to this purpose, centered at grid points, would result in the selection of the closest grid points (ray \( (s_{i+1}, u_p) \) to the right of the reddish ray \( (s, u) \) in Figure 2.8). A bi–linear hat basis in the \( u \) and \( v \) dimensions and a box basis in \( s \) and \( t \) causes bi–linear interpolation in \( u \) and \( v \) and closest grid point selection in \( s \) and \( t \). The beige area Figure 2.8 b) visualizes the support of this choice of filter kernel. Likewise, Figure 2.8 c) shows the area of support for a quadra–linear basis, that causes bi–linear interpolation in all four dimensions. This quadra–linear basis is exactly the one used in the Lumigraph system which has been labeled \( B_{i,j,p,q}(s, t, u, v) \) before.

However, some knowledge of the geometry of the object of interest may be used to help define the shape of such basis functions, i.e. to perform a depth correction as is pointed out in detail in [45]. Assuming a box basis as interpolation filter kernel, and given the situation shown in Figure 2.8, the ray \( (s_{i+1}, u_p) \) would be chosen as being the closest one to the requested ray \( (s, u) \). When considering a hypothetical setting with some object as is shown in Figure 2.8 d) it is obvious that there might be rays that contain values a lot
closer to the true value, for instance ray \((s, u_p)\) —simply because such a ray intersects the object's surface closer to the true intersection with the requested ray \((s, u)\). Supposed depth information of the object is available, the desired intersection point of a given ray \((s, u)\) with the object's surface gets computable. In other words, using depth information and applying the theorem on intersecting lines, one can compute the exact intersection of rays with the \(uv\)-plane starting at \(s\), for instance, and passing through the exact same geometric location on the object's surface as the requested ray \((s, u)\) does. As a result, grid point \(u_p\) gets remapped to \(u'_p\).

The remapped, i.e. depth-corrected \(u\) and \(v\) values may be used to adapt the shape of the basis functions accordingly. The remapping of \(u\) and \(v\) to \(u'\) and \(v'\) performs this reshaping. The new basis function \(B'\) is defined by first finding \(u'\) and \(v'\) and then evaluating \(B\), as is shown below:

\[
B'_{i, j, p, q}(s, t, u, v) = B_{i, j, p, q}(s, t, u', v')
\]

As a matter of course, the depth-correction procedure as a whole requires the object's geometry to be known. Since the only input available is a set of images, 3D shape information must be extracted from these images—a classical focus of computer vision research. The algorithm employed in the Lumigraph system is the octree construction procedure, a so-called space carving method described in [126]. Each input image is first segmented into background and foreground, i.e. the object itself, using a blue-screening technique, as is shown in Figure 2.9 a). The object is then successively carved out from an initial octree representation of a cube enclosing the object. This is done by projecting the voxels of the octree representation into each segmented image. By testing the voxels against the contour in each image, a successive subdivision of the octree is possible, which results in a crude 3D shape approximation by a collection of small voxels, as is visualized in Figure 2.9 b). The external polygons of it are collected in order to form a closed object surface, which is then smoothed using a polyhedral smoothing algorithm presented in [128].
Comprehension (Light field rendering). As is well-known in the meanwhile LF like rendering systems take a large amount of input images, and, consequently, they consume a large amount of memory for their internal representation (Table 1.1 on page 4). In order to make such representations practical they must be compressed.

The processing pipeline of the Light field rendering system does include a compression stage. The Lumigraph system does not include compression at all, but lists it as an important direction for future work. However, the compression scheme included in the former system is a two stage strategy. The first step performs a lossy compression by applying a fixed-rate vector quantization [42] (VQ), whereas the second step involves an additional lossless entropy coding using Lempel–Ziv coding [147].

The codebook is generated from a subset of the Light field representation in order to keep the costs of training small. Small 2- or 4-dimensional tiles of the Light field function are used as codewords. The compression rate achievable using this scheme is reported to be 24:1. After vector quantization, both the codebook and the set of indices are compressed using gzip, a widely used implementation of the Lempel–Ziv coding algorithm. This second stage gives another 5:1 compression, which results in a total of 120:1. Following this two-stage compression scheme, decompressing does also proceed in two separate steps, of course. Compressed data must first be decoded by again applying gunzip and is then ready for any random access using the list of indices as lookup table.

The most important advantage of vector-quantized data is fast random access as data is being decoded, while providing reasonable compression gains. Disadvantages are the very time-consuming preprocessing, i.e. the generation of the codebook from the training data, and the size of the codebook that has to be set manually. However, other compression schemes successfully employed in the field of IBMR will be discussed in Section 2.3.3 whereas the custom compression procedure tailored to the needs in the context of this thesis will be introduced in Section 6.2.

Despite all differences that have been discussed, Light field rendering and the Lumigraph still are strikingly similar IBMR methods. Marc Levoy, one of the authors of Light field rendering, comments on these analogies in [73]. Furthermore, a brief discussion of which idea originated where as well as information about the terminology chosen is also given.
Follow-up Work

Since both Light field rendering and the Lumigraph have been presented in 1996, the field of IBMR has received even more attention than before. Accordingly, a great variety of extensions and improvements in terms of reconstruction quality, memory needs, etc. has appeared. Some of this follow-up work closely related to the original Light field rendering and Lumigraph publications will be reviewed below.

**Dynamically reparameterized light fields** [63]. This extension to the Light field technique introduces dynamic reparameterization as a means to deal with significant and unknown depth—variation in a 3D scene, yet without requiring approximate scene geometry. It thereby gives nice solutions to a number of shortcomings of the Light field rendering and the Lumigraph method together.

Light fields as presented in Light field rendering are only suitable to deal with scenes of approximately constant depth—if not excessively high sampling rates are applied. On the other hand, the Lumigraph technique uses depth—correction to handle scenes with greater depth variation. This requires an at least approximate knowledge of scene geometry that may be hard to acquire. Additionally, both methods are fixed—focus systems, i.e. for a given ray query both systems will produce a single result which will always be the same.

In order to overcome all of these deficiencies a very flexible ray parameterization is defined that consist of a camera surface and a focal surface. The former one is in function identical to the front plane of the standard 2—plane parameterization. The latter one, the focal surface, is a dynamic 2—dimensional manifold. Establishing a mapping that tells which original data camera rays intersect a specific point on a specific focal surface allows for a series of nice effects, that are variable focus, variable depth—of—field through variable aperture and even multiple focal surfaces.

**Surface light fields for 3D photography** [138]. The term *Surface light field* has been created in an earlier publication [91] where it has been defined as a function assigning an RGB value to every single ray leaving every point on a surface. The work presented in [138] is based on this representation. It introduces a complete framework for construction, compression, interactive rendering and simplest editing of surface light fields of real objects.

In contrast to Light field rendering and the Lumigraph, the approach depends on both high—resolution geometry—acquired using a laser range scanning device—and a dense set of image samples which need to be registered with the geometry mesh. On the other hand, it removes the convex hull restriction of the standard 2—plane parameterization employed in both Light field rendering and the Lumigraph, and it produces sharper renderings while better compressing comparable input data sizes.

The best compression ratios are achieved by applying a generalized form of a principal component analysis on the high—dimensional space of *Lumispheres* whereas a generalized form of a vector quantization yields visually more accurate results. A Lumisphere, however, is introduced as a piece—wise linear RGB—valued function with the help of which surface light fields may be defined. Both compression strategies generate a small set of prototype lumispheres, representing each lumisphere as a weighted sum of such prototypes; either as an index as is done in vector quantization, or as a set of coefficients as is done in principal component analysis.

Note that the construction of all Lumispheres, i.e. the assembling of input images into samples of lumispheres, each consisting of a color and a direction value, is a resampling
problem. This technique thus introduces an additional sampling step to the construction of the internal representation, comparable to the re-binning step of the Lumigraph system (Section 2.2.3) as is also stated in [12].

A technique similar to surface light fields that does not require resampling of input images is view-dependent texture mapping (VDTM) [35, 36, 102] which may be understood as a kind of a light field. Input images are reprojected into desired camera views, thereto using geometrical information. However, a surface light field may be seen as a more efficient representation of view-dependent texture mapping as is noted in [36].

Plenoptic sampling [20]. A very important question arising in the context of IBMR is how many samples of the Plenoptic function and how much geometrical and textural information are needed to generate a continuous representation of the Plenoptic function. Plenoptic sampling answers this question; moreover, it addresses the problem of appropriate sampling rates for anti-aliased light field rendering. This minimum sampling rate is mathematically derived from a spectral analysis of light field signals using the sampling theorem.

The author state the problem of light field reconstruction as the problem of finding a reconstruction filter for anti-aliased light field rendering, given the sampled signals. This reconstruction filter is understood as a combined filtering and interpolating low-pass filter.

The analysis of the spectral support of light field signals is first established by considering the simplest scene model in which every point in space is at a constant depth \( z_0 \). If some information about scene geometry is known, more complex scene geometry can be decomposed into a collection of these constant depth models on a block-by-block basis.

The findings presented are significant. Firstly, the analysis of minimum sampling rates is based on the fact that the spectral support of light field signals is bounded only by the minimum and maximum depths of a scene, no matter how complicated the spectral support might be because of depth variations in the scene. Given this minimum and maximum depths in a 3D scene, a reconstruction filter with an optimal constant depth can be designed for anti-aliased light field rendering. The optimal constant depth \( z_c \) is derived from the minimum and maximum depths and it is consistent with the scene’s average disparity \( d_c \) as is shown in Equation 2.5 below.

\[
d_c = \frac{1}{z_c} = \left( \frac{1}{z_{\min}} + \frac{1}{z_{\max}} \right)/2
\]  

(2.5)

The best rendering quality is reported to be obtained at the optimal depth \( z_c \) and not at the focal plane as it has been commonly assumed before. Consequently, the optimal depth is suggested to be used as a guidance for selecting the focal plane.

Furthermore, the analysis of the bounds of a light field’s signal allows to analytically compute the minimum sampling rate for light field rendering. The reader is referred to [20] for any formula. Similar results may be found in [40] where a detailed optical analysis of light field rendering is presented, as well as in [78] where a geometrical approach is being followed.

Finally, the minimum sampling problem is also studied in the joint geometry and image space. The findings confirm the general understanding of the trade-off between the amount of geometry and the amount of imagery used in an IBMR method. As more geometrical information becomes available fewer images are necessary, given any rendering
resolution. (Again, the reader is referred to [20] for visualizations of the resulting curves.) This fact has already been visualized in a very similar way by the Rendering spectrum in Figure 2.2. Adding geometrical information to a purely image-based system lessens the need for a large number of input images and vice-versa. The lumigraph technique in fact delineates a method taking advantage of knowledge of scene geometry and therefore asking for less input images for a comparable output quality when compared to Light field rendering.

**Unstructured lumigraph rendering [12].** There is more than one way to uncover the connection between IBMR methods making usage of geometrical information and methods that do not. Plenoptic sampling, as has been discussed above, exploits an analytical, mathematical point of view. In contrast to that, Unstructured lumigraph rendering (ULR) bridges between these two sides using an algorithmic approach. Additionally, it generalizes many IBR methods into one framework, including light field and lumigraph style rendering and VDTM, thus allowing for unstructured input and taking advantage of an approximate scene geometry, if available.

In contrast to many previous techniques, ULR does not require to re-sample any input images but stores the original pictures. This is due to the absence of a ray parameterization, say the standard 2-plane parameterization in particular. The input images may be acquired using a hand-held video camera or 3D digitizing devices for indoor scenes. Thus, the input to the ULR algorithm is a possibly unstructured collection of original source images with their corresponding pose information plus a **scene proxy**, i.e. approximate geometric information.

The ULR algorithm directly renders new views from the input set by evaluating a **camera blending field** that is used to combine the acquired source views. In particular, the blending field describes how each source image is being weighted to reconstruct a given pixel in the new view.

The calculation of this blending field is based on a set of desirable properties for any IBR algorithm. Table 2.2 gives an overview of the list of these eight properties and opposes some IBR methods using this list. The reader is referred to [12], Table 1, for a wider comparison and also for an in-depth discussion of the properties themselves. The PMIS method, Plenoptic modeling and rendering from image sequences, [56] will be discussed in Section 2.3.1.

The actual rendering of novel views includes the following steps. After a sparse set of sample locations for the camera blending field are being selected, these points are triangulated. The camera blending field is then evaluated and normalized at the grid points only and interpolated over the rest of the new view. To this end, texture information from involved source views is being projectively mapped onto proxy triangles with the corresponding blending values set in the alpha channel.

**Scam light field rendering [144].** A Scam—an abbreviation for surface camera—is a means of a new ray parameterization, not unlike the one used for the SLF technique [138] where rays are parameterized over the surface of a scanned geometry. A scam is a data structure that collects the set of all rays from a light field which are directly and indirectly associated with a 3D point correspondence. It can therefore be understood as a pinhole camera, anchored at a given 3D point correspondence, reversely oriented compared to the set of input cameras, often sitting at a surface point of a 3D scene.
TABLE 2.2 Comparison of different IBR techniques using the set of properties defined in ULR [12].

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Use of geometric proxy</td>
<td>—</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2) Epipole consistency</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3) Resolution sensitivity</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>yes</td>
</tr>
<tr>
<td>4) Unstructured input</td>
<td>—</td>
<td>resampling</td>
<td>yes</td>
<td>resampling</td>
<td>yes</td>
</tr>
<tr>
<td>5) Equivalent ray consistency</td>
<td>yes</td>
<td>yes</td>
<td>—</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>6) Continuity (color)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>7) Minimum angular deviation</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>8) Real-time rendering</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

The basic idea in Scam light field rendering is to factor all rays of a light field into two separate classes. Rays that are consistent with a given correspondence are collected and represented as a scam. All remaining rays are described using the standard 2-plane parameterization. 3D correspondences are generated using different stereo algorithms, such as described in [13] and [38]. A user of the system has the possibility to specify image regions where correspondences should be established, in order to significantly improve reconstruction quality in specific areas of interest. The system only needs a sparse collection of established correspondences in order to perform the ray factorization. It may therefore be placed between LFR based on a light slab representation without any geometric knowledge and the LUM system which utilizes a continuous yet approximated proxy geometry.

For the purpose of novel view generation, i.e., rendering, all contributing scams must first be projected into the desired view. The image is then synthesized from these scattered projection points using a color blending approach similar to the camera blending field of ULR. The main idea is to blend projected correspondences, which are assumed to be relatively accurate, in a 1-pixel-ring neighborhood only. Projected data is being blended against data directly reconstructed using standard LF rendering with the focal plane optimally placed at the depth associated with the average disparity as is suggested in [20].

The novelties and extensions of this thesis may also be related to the original LFR and LUM publication, as has just been done with selected, other follow-up work.

The setup presented basically follows the LFR approach, that is, no geometrical knowledge is assumed. On the other hand, contrary to LFR, two alternative parameterizations are explored instead of the 2-plane light slab setting. Furthermore, unstructured input data can also be handled in pretty much the same way as is done in the LUM system. That is, the re-binning step as well as a scattered data interpolation extension are borrowed and extended. However, emphasis is put on the hierarchical representation of light field data as well as on the operators necessary to process thus coded, dynamic data.

An overview of all relevant system components as well as more detailed information on their relation to LFR and the LUM are given in Chapter 3.
2.3 STATE OF THE ART

Besides the pioneering work and all follow-ups in the field of LF like image-based rendering systems, there are numerous further publications which are important in the connexion of this thesis. Each of these publications emphasize a certain aspect in the field of IBMR techniques. However, they might be divided into four groups according to their focus, that are, data acquisition, data parameterization and data compression. The fourth group contains methods applying wavelets to various aspects of IBMR, e.g. hierarchical representations, compression, etc.

The following sections give a compilation of selected previous work separated into these four groups.

2.3.1 Acquisition

When talking about acquisition in the field of IBMR, a few things must be distinguished between. Firstly, it is not only about acquisition of image data. Quite a few hybrid IBR systems also have to deal with the problem of acquiring geometric data. Secondly, when it comes to camera pose estimation of acquired real life imagery, there are two fundamental techniques to be followed. Either, some computer-controlled equipment that allows to accurately measure position and orientation is used as is done in LFR [75], or, the same information is directly extracted from the acquired images using well-known and available calibration procedures (such as [129, 146, 58] for instance) as is done in the LUM system [45]. However, an uncalibrated image acquisition system has also been shown as is discussed below. Lastly, the image acquisition equipment may be compiled more or less elaborate and more or less costly, ranging from one consumer camera over a whole array of cameras to custom-made acquisition hardware.

The acquisition of scene geometry in case of virtual 3D scenes is trivial, since models are readily available. Concerning real life scenes, the same distinction as in case of image acquisition may be applied. Either, geometry is being read using special, computer-controlled equipment such as laser range scanners as is done in the SLF system [138], or, 3D shape information is tried to be recovered from acquired imagery using computer vision approaches as the LUM system [45] does.

However, the problem of geometry acquisition will not be further deepened since it is not a relevant topic in the context of this thesis. Acquisition of image data will be briefly reviewed according to the issues stated above.

Computer-controlled image acquisition. As has been mentioned and discussed before, the image acquisition setup of the LFR system is a construction based on a motion platform and additional stepping motors that allow to translate, pan and tilt a mounted camera, thus providing accurate pose information to the controlling computer. An illustration has been shown in the left image of Figure 2.7 on page 21.

Huge, heavy and costly equipment was used in the context of the Digital Michelangelo Project [74, 76]. Images of the planar gantry and the mounted camera were shown in Figure 1.1 on page 3. As a matter of course, this impressive equipment was not only designed for the acquisition of light field data. It was rather planned as a 3D scanning device for large statues using laser range and digital image sensors, with the acquisition of a large light field of Michelangelo's statue of Night as a side project.

Rendering with Concentric mosaics [115] presents an alternative parameterization for light fields, that decreases the dimensionality of the data representation by giving up the
vertical parallax. To this end, camera motion is constrained to planar, concentric circles which gives a natural ray parameterization using the three parameters radius, rotation angle and vertical elevation. However, capturing Concentric mosaics only involves a single, off-centered camera that rotates along a circle. Rotation angles are known using a rotary table.

By using radial and angular information plus a ray’s elevation as parameterization indices, the concentric mosaics may be classified as a 3–dimensional representation for samples of the plenoptic function (PLF). In fact, an alternative categorization of IBMR methods suggested in [115] follows the dimensionality of the representation for plenoptic samples as is shown in Table 2.3.

The image acquisition system used in the context of the Surface light fields (SLF) [138] moves a camera over a sphere. The spherical gantry thereby gives the positional information of sampling locations relative to one another.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Viewing space</th>
<th>IBMR method</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>7D</td>
<td>free</td>
<td>Plenoptic function (PLF) [2]</td>
<td>1991</td>
</tr>
<tr>
<td>5D</td>
<td>free</td>
<td>Plenoptic modeling [89]</td>
<td>1995</td>
</tr>
<tr>
<td>4D</td>
<td>inside a 3D box</td>
<td>Light field rendering (LFR) [75]</td>
<td>1996</td>
</tr>
<tr>
<td>3D</td>
<td>inside a 2D circle</td>
<td>Lumigraph (LUM) [45]</td>
<td>1999</td>
</tr>
<tr>
<td>2D</td>
<td>at a fixed point in space</td>
<td>Concentric mosaics [115]</td>
<td>1999</td>
</tr>
</tbody>
</table>

**Custom image acquisition hardware.** A logical extension to computer–controlled, single camera acquisition setups is a system with multiple video cameras, arranged in an array–like manner, thus saving the need for precisely moving and positioning the camera, but, on the other hand, at the expense of a multiple of hardware needs.

The Distributed light field camera [142] presents a system that allows multiple users to navigate in a dynamically changing light field that is captured in real–time. The light field camera consists of 64 low–cost consumer video cameras that are connected to standard PCs through FireWire. This design relaxes the constraint of many IBR techniques that are only capable of capturing and displaying static scenes. Hence, the PLF being reconstructed by the distributed light field camera is actually 5–dimensional (Equation 2.1 on page 16) while the representation at a single moment in time is still 4–dimensional.

Several IBR systems using multiple video cameras have been demonstrated, thus allowing the display of real, evolving 3D environments. The reader is referred to [142] for a short overview.

The Light field video camera [135] goes one step beyond the assembling of consumer video cameras that form an array by arranging a battery of custom nodes each of which containing a CMOS image sensor and an MPEG–2 compression unit, controlled by an FPGA chip. The light field video camera is designed to record synchronized video data from a large array of such nodes. While in [135] only a six–camera prototype has been
demonstrated, the Stanford multi–camera array [21] shows a few possible setups using up to 128 cameras. However, the light field camera is mainly targeted at compression and storage of recorded data, and not at real–time interactivity.

An exciting alternative to video cameras playing the role of the digitizing element in an acquisition setup has been shown in [141, 143] where a readily available, low–cost flatbed scanner is used instead. The primary motivation for this rather unusual idea was to design a cheap capture device for light field data that is also transportable and therefore usable for acquiring outdoor scenes. The scanner's glass is equipped with an 8–by–11 grid assembly of one-inch plastic lenses. While the construction of such a device is rather simple, processing the scanner's raw output imposes dificile challenges, such as color correction and radial distortion compensation.

**Hand–held camera solutions.** An IBR technique that accepts structured and unstructured input, i.e. images captured with computer–controlled equipment as well as with hand–held cameras is the ULR [12] system, as has been previously mentioned.

The Lumigraph [45], however, uses a simple acquisition stage with cheap equipment, namely a single hand–held video camera. While this is a non–costly solution in terms of material and apparatus, it does not exactly hold in terms of algorithmic cost, as has been discussed earlier. Furthermore, input image calibration relies on dedicated calibration markers in the capture stage as is apparent in the right image of Figure 2.7 on page 21.

There are ways to calibrate image data without the need for dedicated calibration patterns or even pre–calibrated acquisition systems, as has been shown in [57]. The method calibrates camera views of extended image sequences, i.e. it recovers structure from motion, with the goal of light field reconstruction already in mind. It is mainly based on the factorization method presented in [123] which provides a closed–form solution if all projections of all scene points into all views are known. The authors of [57] extend the original method for long image sequences, especially when scene points appear and disappear, which is normally the case for natural camera movements. This extension that mainly applies the factorization method of [123] to parts of the sequence and merges the results together is tested and evaluated on simulated data.

PMIS [56] implements an IBR system that renders arbitrary new views from long image sequences, captured by a hand–held video camera and calibrated using a very similar structure–from–motion approach. New views are generated directly from the calibrated sequence of recorded images with the use of depth maps, which are recovered locally for each view point by applying a stereo matching technique [69]. The system as is described in [56] stresses the mapping of all given input images to a single plane that approximates the scene's geometry—which resembles the $uv$ parameterization plane of the LFR [75] technique. A refinement using a mapping via local planes for each input view based on depth information greatly enhances the quality of renderings. A very similar publication by the same authors [70] further emphasizes the structure–from–motion approach and describes how to considerably improve the calibration by exploiting the 2D topology of input camera view points.

However, the local planes may be seen as an approximate scene proxy bringing in geometrical knowledge about the 3D scene. Furthermore, by directly storing the input views an additional re–sampling step can be omitted, which makes the approach very well suited for unstructured input, and, which makes it a technique very similar to ULR [12] as is also reflected in Table 2.2 on page 28.
Uncalibrated image acquisition. A fully calibrated acquisition stage with known intrinsic and extrinsic camera parameters is normally precisely defined—up to a scaling factor. This is not necessarily required, though, as it has been shown in [114] where an uncalibrated light field acquisition system is described.

In lieu of establishing the mapping of image points from a camera's image plane onto the light field's parameterization planes with the help of calibrated camera parameters, all that is computed is the projection of the camera's image plane onto the parameterization planes. Since the standard light slab setting is being employed, the projection is a simple linear projective transformation, a so-called 2-dimensional planar homography [54]. Hence, the mapping of points on the image plane onto points that lie on the planes of the light slab may be expressed via a single projection matrix.

The capture stage of the acquisition system described in [114] physically replicates a single light slab. That is, an object is positioned between two parallel planes, a transparent front plane and an opaque back plane, both of which with four marker points on them. Input imagery is captured moving a hand-held video camera in front of the capture setup. The homographies between every single input image plane and both parameterization planes can be computed by identifying the four maker points on each plane. This four point correspondence yields enough equations to solve for all unknowns of the projection matrix. Thus established homographies are then applied to image pixels in order to project input image data into the light field representation.

The rather simple computations involved make the system suitable for on-line light field acquisition. Furthermore, refinement operations at badly covered or even undefined regions of the light field are feasible, too.

Albeit such nice properties, the method has its limitations. A severe drawback is the fixed, physical setup of the parameterization planes, which makes it impossible to place the parameterization's back plane—the focal plane—at the optimal depth, as is suggested in PS [20].

Adaptive acquisition. As has been discussed before, PS [20] addresses the questions of how many samples of the Plenoptic function, i.e. input images, are needed to generate a continuous representation of the Plenoptic function. The results, i.e. the minimal sampling rates for anti-aliased light field rendering, are being derived from a spectral analysis of light field signals. However, the problem of how to identify and acquire the thus characterized minimal set of input images has not been tackled. Interestingly, a few publications have addressed this question before Plenoptic sampling [20] has been presented, all of them following different approaches.

In [117] viewpoints of a lumigraph are organized on an adaptive triangle mesh in order to control the number of viewpoints and, thus, the amount of memory needed used for reconstruction. An initial regular and fine mesh of viewpoints is being coarsened by neglecting viewpoints that contribute little to the reconstruction quality of the lumigraph. The measure adopted is a mixture of benefit and cost of incorporating a specific viewpoint.

While such an adaptive data selection scheme makes perfect sense in terms of viewing, it does not in the context of image data acquisition. Other publications, in contrast, focus on adaptiveness during data acquisition in particular.

A warping-based refinement of lumigraphs has been described in [55] where an initially sparse lumigraph data set is continuously refined through the use of a specialized
It takes the closest images from the lumigraph and performs a warping operations to some new viewpoint which results in a new input image for the lumigraph. The warp itself requires geometric information, i.e. depth values, which is assumed to be available.

A follow-up publication by the same authors extends these methods to an adaptive acquisition system of lumigraphs (AAL) [107]. The system decides which new views of a scene should be rendered on demand and added to the light field. The prediction of potential improvement in image quality when adding such a new view is based on the warping algorithm presented in the former publication. In other words, the system predicts how well the image for a candidate viewpoint can be reconstructed by means of warping a subset of already acquired images.

The work presented in [107] concentrates on how to estimate a reconstruction error from color and range image data. The suggested metric determines a scalar per-pixel error as a weighted sum of 1) a blending error which accounts for deviations of warped pixels from the final color, 2) a hole error which considers holes in the destination image where no source pixel maps to and 3) a single map error that takes destination locations into account with only one source pixel mapped to.

Based on this error metric, the system decides whether it is necessary or not to acquire a new image for a candidate viewpoint by rendering and inserting it into the lumigraph representation. Any new image is rendered using a ray-tracer, therefore range data is available for free. As a consequence, the system is confined to synthetic scenes only.

The acquisition component of the system as is described in this thesis may be contrasted with the issues about image data acquisition that have just been discussed.

Because the system is primarily designed for future use with a hand-held camera, it accepts unstructured input just like ULM [12] and PMIS [56] as well as LUM [45] and SLF [138] through re-sampling do. Moreover, it is constructed as an on-line acquisition system—unlike PMIS—that refines an initial set of input images, pretty much in the spirit of the adaptiveness in AAL [107]. As a matter of course, having the video stream from a hand-held camera in mind, the system as is presented in this thesis is not confined to synthetic scenes only.

However, aiming at a rapid on-line acquisition system, extrinsic camera calibration cannot be accomplished as is done in PMIS, for example. Instead, what becomes necessary is an external, independent tracking mechanism in order to estimate the camera’s pose. As is explained in detail in Chapter 7, the system has been evaluated using a motorized object rig with a camera attached to it. This construction is conceptually able to cover a full sphere around an object at its center. Such a computer–controlled acquisition system that delivers real-world imagery to the core system represents a passable compromise to a hand-held camera for evaluation purposes.

The succeeding Chapter 3 discusses the design of the acquisition component as well as the interaction with the system’s core in greater detail.

### 2.3.2 Parameterization

LF-like rendering systems are based on the ray database, a collection of samples of the PLF that describes the directional radiance distribution for every point in space. The
FIGURE 2.10 Possible settings for the parameterization of oriented lines in 3D space.

a) Point and direction (PDP, DPP parameterization).
b) Two points on a sphere (2SP parameterization).
c) Points on two planes (2PP parameterization).
d) Original images and their camera pose.

subset of that function in an area free of occluders surrounding a 3D scene can be represented in only four dimensions (Section 2.1, Figures 2.5 and 2.6).

The support of this 4-dimensional light field function is the set of oriented lines in the 3-dimensional cartesian space. Thereupon, a representation of this support asks for a 4D line parameterization.

It is, however, quite easy to understand why the set of oriented lines in 3D is a 4D space. Any oriented line can be uniquely described by its direction and its intersection with the unique plane orthogonal to its direction that passes through the origin of a world coordinate system. Any direction may be represented by two parameters, an azimuth and an elevation, with respect to the world coordinate system. The intersection point with the plane may be represented by its cartesian coordinates with respect to a local coordinate system attached to the plane, yielding the additional two parameters.

The representations of the two pioneering publications in the field of IBMR, the model presented in LFR [75] and the model introduced in LUM [45], share both the same parameterization as was already pointed out in Section 2.2.2. Both models are based on the so-called light slab, the standard 2-plane parameterization (2PP) that represents a line in space by its intersection points with a pair of parallel planes.

Most of the follow-up work is based on the 2PP, too. As is reported in [16] this choice of parameterization was primarily inspired by traditional 2-step holography [8, 53]. However, the main reason for its wide and popular use is its computational simplicity that avoids cylindrical or spherical projection calculations. Moreover, it permits to take advantage of the texture mapping subsystem of any graphics hardware during rendering. While this was by far the fastest way of rendering light fields in the late 90's, nowadays CPU speeds are swift enough to achieve real-time performance with pure software solutions, too. Therefore, not using the 2PP does not entail a severe penalty in terms of rendering efficiency anymore.

The structure of the 2PP has been explored in detail in [50]. In particular, the relation between the geometry of the scene and subsets of the 4-dimensional data is analyzed. This knowledge is interesting theoretically but is also potentially important for a variety of applications, e.g. compression. However, emphasis is put on the basic theoretical issues.

As a matter of course, parameterizing oriented lines in 3-space cannot only be accomplished with the light-slab setting. Figure 2.10 above shows a few more alternatives, e.g.
2.3 STATE OF THE ART

Figure 2.11 Polar parameterization for oriented lines in 3D space.

A parameterization using (a) a point and a direction or (b) two points on a sphere. The setting in Figure 2.10 c) shows the standard 2PP. In principle, intersections of an oriented line with any 3-dimensional basic shape or any reasonable combination of 3D basic shapes, such as planes, spheres, cylinders, etc. may be used to parameterize the space of oriented lines in 3D.

Figure 2.10 d) illustrates the representation employed in ULR [12] and PMIS [56], for instance. It may rather be understood as a kind of image database than as a ray database since the original input images are stored, along with their pose information.

Moreover, using local coordinates of intersection points of an oriented line with some 3D basic shape is not the only fundamental way of parameterizing the set of oriented lines in 3D space. As an example, an alternative, novel parameterization for light field data is presented in [130] where oriented lines, the rays, are uniquely identified by the four polar coordinates \((r, \phi, \theta, \omega)\). The point located on some ray with shortest Euclidean distance \(r\) from the origin of the world coordinate system defines the three polar parameters \(r, \phi, \theta\) in the usual manner. The remaining parameter \(\omega\) is obtained as the rotation about \(\mathbf{l}\) from \(r \times \mathbf{l}\) to \(r\) as is shown in Figure 2.11 above.

In contrast to the 2PP, the polar parameterization as is described in [130] does not require any explicit partitioning to cover the whole surrounding of an object. Furthermore, it offers the flexibility to adaptively sample to the edges of an object. On the downside, it fails for rays that intersect the origin as well as for rays that are characterized by \(\phi = \pm(\pi/2)\). The former limitation is due to \(r = 0\) in that case, the latter because of an undefined angle \(\theta\). However, it has been shown that the polar parameterization is likely to produce visually more pleasing results than the 2PP.

Two more alternatives have been presented simultaneously with the polar parameterization. In [17] the 2-sphere parameterization (2SP) as well as the sphere-plane parameterization (SPP) have been introduced to the field of LF rendering. These two parameterizations solve the disparity problem that is caused by a non-uniform parameterization of the line space. The standard 2PP suffers from this problem, since the density of rays induced by the 2PP is biased towards certain directions and the spatial sampling is different for each direction. (The reader is referred to [17] for any figures.) Artifacts
caused by the disparity problem are obvious when a reconstructed image uses samples from more than one light slab.

Both parameterizations, the 2SP and the SPP, are based on random processes which uniformly sample the space of oriented lines that intersect a tight sphere surrounding an object. Therefore, the techniques presented in [17] do not inherit sampling biases induced by the parameterization and they provide uniform image quality from any angle of view.

The 2SP parameterizes an oriented line by its intersections with two spheres. Typically, both spheres are identical and hence coincide with the virtual, tight ball surrounding an object. The SPP parameterizes rays by their intercept points with a sphere and a plane intersecting the sphere. Again, the parameterization sphere tightly fits around an object.

Unbiased or isotropic parameterizations like the 2SP and the SPP own the property of uniformity due to their construction, i.e. their implementation is invariant under rotation and translation [17]. However, one might argue that this notion of uniformity is too weak, that light field parameterizations should additionally be invariant under perspective projections.

In [18] (see [16] for a summary), 4D light field parameterizations and their suitability for LF rendering are studied, i.e. the 2PP, the 2SP and variations of the SPP which are called direction–and–point (DPP) and point–and–direction parameterization (PDP). Firstly, light field uniformity is formally defined as statistical uniformity in the continuous domain and as sampling uniformity in the discrete domain. Hence, statistical uniformity applies to continuous light fields and it is shown that this is a prerequisite for sampling uniformity and the construction of uniform LF discretizations. However, among the analyzed parameterizations, only the 2SP and the DPP satisfy the statistical uniformity property.

Uniform light field models still lack an important property of geometric models, which is the view independence. This property guarantees that a model is not only invariant under rotation and translation but under perspective projection, too. It is argued in [16, 18] that since this feature is usually expected of a geometric model it should also be expected of a radiometric model. As a result it is shown that, unfortunately, none of the most common LF parameterizations owns the view independence property. On the other hand, among all parameterizations studied, the DPP introduces the least number of biases at rendering time—and is therefore the parameterization of choice.

In contrast to these very tangible results, a purely theoretical but not less interesting concept is proposed in [44]. The rayset is a general, higher–level notion that may be used to classify and compare all existing IBR techniques, mainly by means of their different parameterizations and sampling schemes employed.

As is defined in [44] the rayset is a parametric function built upon \( U \), a parameter space, and \( T \), an attribute space such as a color space, possibly enhanced with additional attributes such as depth, normal information, etc. The formal definition is given in Expression 2.6 below.

\[
\begin{align*}
(x, y, z, \theta, \phi) &= S(U) \\
T &= A(U)
\end{align*}
\]
The function \( S(\mathbf{U}) \) is a support function that maps from the parameter space to the ray space whereas the function \( A(\mathbf{U}) \) represents an attribute function, mapping from the parameter space to the attribute space. The rayset explicitly separates the mapping from the ray space to the attribute space into two separate mappings, using these two mapping functions. It thus introduces additional flexibility by means of changing scene objects or scene distortion depending on which function is being changed independently.

The parameterization component of the system as is described in this thesis offers two different, 4-dimensional parameterizations to be used. Both of them parameterize the set of oriented lines in 3D space by their intercept points with basic 3D shapes. The first of which is called a cylinder–plane parameterization, the second a sphere–plane parameterization which is very similar to the SPP [17] and the DPP [16, 18] respectively. As was mentioned before, both parameterizations will be introduced in Section 3.2 and discussed in greater detail in Section 6.1.

Table 2.4 below resumes a few key properties of parameterizations mentioned in this Section 2.3.2. It already lists the cylinder–plane and the sphere–plane parameterizations, too.

<table>
<thead>
<tr>
<th>Parameterization</th>
<th>Ref.</th>
<th>Intersection-based</th>
<th>Disparity problem</th>
<th>Uniformity</th>
<th>View independence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2PP</td>
<td>[45, 75]</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Polar</td>
<td>[130]</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>SSP</td>
<td>[17]</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>2SP</td>
<td>[17]</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>PDP</td>
<td>[18]</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>DPP</td>
<td>[18]</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Cylinder–plane</td>
<td>[82]</td>
<td>yes</td>
<td>no (^2)</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Sphere–plane</td>
<td>[82]</td>
<td>yes</td>
<td>no (^2)</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

1 Introduces the least number of biases at rendering time.

2 In case of uniform sampling.

### 2.3.3 Compression

Most image–based methods are built on high–dimensional structures for their internal data representation (see Table 2.3 on page 30). LF–like models in particular require a 4D construction to store their database of rays. As a consequence, IBR systems most often need to handle a vast amount of data and hence, compression is a vital topic to be addressed.
Although the amount of data to be stored grows with respect to the resolution of the individual dimensions of the multi-dimensional data set, there is a good chance to obtain high compression ratios without significant loss of quality due to the highly correlated values in each dimension. However, there exist a number of different strategies to be followed in order to efficiently compress image-based or light field data in particular. Which scheme works best is dependent on an actual problem setting, of course. A short list of reasonable compression strategy is given below.

1) **Input compression:**
   Compression of individual input images through the usage of well-known image compression techniques.

2) **Video compression:**
   Adaptation of video compression techniques, e.g. reference coders, to \( n \)-dimensional arrays of images, with \( n = 2 \) for LF-like models.

3) **Block coding:**
   Decomposition of the high-dimensional data set into blocks of lower dimensionality with subsequent encoding using vector quantization, for example.

4) **Transform coding:**
   Decomposition of the data set as a whole into some basis functions with subsequent processing and encoding of the resulting coefficients.

All of the above mentioned schemes have been tried as is reported in the literature. A short but fairly comprehensive overview may be found in [98]. A few of these approaches will be briefly reviewed in the following sections.

**Input compression.** A light field coder that exploits inter-image similarities by compensating disparity between images has been presented in [85]. The coding process starts with intra-coding of a small subset of input images, using a block-based *discrete cosine transform* [3] (DCT), a standard compression technique for still images. Intermediate images are recursively predicted by disparity-compensating multiple surrounding images, thereby building up a hierarchical description of the full light field from the initial small set. Disparity maps are Huffman-coded [59] and predicted images are refined using DCT coding, too, if necessary. It is reported that compression ratios of 1000:1 are achievable at acceptable quality of reconstructed views.

As is mentioned above, the still-image coder adopted works on a block basis. While, in principle, any still-image compression method could be used, the coding process as is presented in [85] may therefore also be classified as a block-coding technique.

However, the same still-image compression technique is adopted in [86] in order to encode a subset of images from the light field's input array, the so-called I-images. Further processing steps are named below.

**Video compression.** The compression technique presented in [84, 86] is based on a video compression codec, modified so to encode the 4D light field structure. The set of DCT coded I-images are selected evenly distributed, thus constituting a subsampled representation of the set of input images. I-images serve as references for encoding the remaining images, the predicted or so-called P-images. All P-images are divided into square blocks of 16 \( \times \) 16 pixels. Each block of each P-image is then connected to one of eight basic operations that combine information from several surrounding I-images, which closely relates this method to multi-frame prediction techniques in video
coding [134]. (The reader is referred to [86] for a detailed discussion of the eight operations.) The remaining data is thus reconstructed by applying these coding modes on all blocks separately. The coder is reported to feature compression ratios in the order of $10^2$ to $10^3$ at medium to high image reconstruction quality.

This coding process clearly adopts methods from video coding. Nevertheless, it may again be classified as a block-based coding technique since it divides the input images into blocks before further processing them.

The same argument holds for another MPEG-like algorithm that compresses lumigraph data. The algorithm presented in [145] is called the reference block coder (RBC).

**Block coding.** The basic idea of block-coding techniques is to chop up an original data set into small blocks of the same or lower dimensionality and then to encode each block individually. The most prominent block-coding compression scheme in the field of IBMR has been presented in the LFR publication [75] itself. A spatial domain vector quantization (SVQ) was used to compress blocks of the light field data set as was reviewed in Section 2.2.3 earlier in this chapter.

As a matter of course, alternative techniques may be used to encode the blocks. In [91], for example, blocks of light field data are compressed using a DCT and Huffman-coding. However, neither of these algorithms achieve as high compression ratios as the methods based on video compression techniques do, because they do not take advantage of the correlation between individual blocks of data.

**Transform coding.** Just as any signal of any dimensionality can be decomposed into coefficients using a set of suitable, weighted basis functions, light field or lumigraph data sets can be transformed, too. Since such data is known as being of high dimensionality, a multi-dimensional transformation must be employed, i.e. a 4D transform if a LF-like data set is being transformed as a whole.

In principle, all transform coders follow the same two-step procedure. Firstly, the signal to be compressed is decorrelated by transforming it using a set of proper basis functions, resulting in a bunch of coefficients. Up to this point, a discrete signal usually occupies the same storage space as the untransformed version, i.e. no compression gain has been achieved yet. In fact, more storage space might be necessary, since coefficients are real numbers that must be stored as floating point values whereas the original signal usually consists of integer numbers of a smaller number of bits.

The second step can be characterized as analysis of coefficients which may involve coefficient thresholding and/or re-quantization, resulting in the actual compression. Thus, the idea of transform coders is to concentrate the signal’s energy in a small number of coefficients through de-correlation. However, a more detailed discussion about various aspects of multi-dimensional transformations and their usage for compression will be given in Section 4.3 and Section 6.2.

As a matter of course, the basis functions of choice are dependent on the exact problem statement. In principle, many options are available. For instance, spherical harmonic functions [32] have been used in [136, 137] in order to attack the compression problem, as was already mentioned in Section 2.2.2. They achieve comparably low compression ratios in the order of $10^1$. 
The wavelet transform, however, is known to gain much better compression ratios. Several publications report compression gains ranging from 30:1 by applying a 2-dimensional transform [61] up to 1000:1 for a fully 4D wavelet transform coder [83].

A detailed comparison among (1) block coders, (2) reference or video coding techniques and (3) high-dimensional transform coders considering the wavelet transformation as an example is given in [77]. The authors conclude that block coders such as the SVQ are the least complex in decoding and can therefore easily achieve just-in-time (JIT) rendering. On the downside, their potential regarding compression is rather poor. Reference coders such as the RBC represent a good compromise between compression efficiency and JIT rendering capabilities. It is further reported that high-dimensional wavelet coders achieve the best compression performance. However, they are also the most complex ones and thus, interactive rendering is tricky to achieve.

The core data in the system as is described in this thesis is projected onto the wavelet domain using a fully 4-dimensional non-standard transformation. As a consequence, compression is achieved through analysis of the resulting coefficients. Thus, the compression sub-system is to be classified as a transform coder.

As was mentioned before the custom compression procedure will be discussed in depth in Section 6.2.

2.3.4 Wavelets

Quite a few publications exploring wavelet theory in the field of IBR methods have been presented since the innovative work of LFR [75] and LUM [45] in 1996. All of them exploit the wavelet transform because of their ability to effectively de-correlate any data and thus provide a powerful framework for compression. However, only a few among them take advantage of other favorable properties of wavelets, too, such as the inherent hierarchical description for instance, that may directly support a level-of-detail feature.

The authors of [71], in contrast, give a whole list of reasons why they propose a wavelet based solution for storing light field data. A few of them are reprised below.

- Benefits with respect to representation:
  The wavelet representation offers a handy tool to control the trade-off between accuracy of reconstructions and storage space needed. Likewise, accuracy may be incrementally ameliorated on a level by level basis. Additionally, a wavelet reconstruction is able to interpolate the original data.

- Benefits regarding computation:
  Error metrics are clearly defined and relatively easy to compute. Moreover, reconstructions of single data items are computationally much cheaper than a full reconstruction. In the setting of [71] the costs for a point-wise reconstruction are reported to be logarithmic in the number of light field samples.

- Benefits concerning compression:
  The wavelet transform exploits the coherence in all dimensions of a data set, if applied accordingly. Hence, a good de-correlation and therefore high compression gains may be expected.
A few representatives with emphasis on the issues toward representation and computation will be briefly reviewed in the following sections. Since transform coders have been briefly discussed in the preceding sections, they will not be further deepened. Besides, every work employing wavelets in the field of IBR also takes advantage of their compression capabilities which makes this topic hard to separate.

**Representation.** The concept of hierarchy using wavelets may be found in several approaches to light field representation and encoding. In [61], for instance, a multiresolution description of the Spherical light field is provided. This term is given by the chosen 4-dimensional parameterization of the PLF [2] which can be understood as a collection of small directional spheres clinging to a larger positional sphere that surrounds a bounded object in 3-space. Sampling a spherical light field means finding a discrete approximation of those two spheres which is done through successive subdivision of initial octahedrons, resulting in triangle-based tessellations of the spheres.

The transformation to wavelet space works on linearized images of the directional spheres, i.e. on reordered triangles of the discretized, directional spheres into 2D arrays. Hence, the whole process is a 2-dimensional transform, using the Haar wavelet, applied in the non-standard decomposition scheme [119]. The encoding process itself achieves a compression ratio of 30:1, after deletion of the smallest coefficients and re-quantization of non-zero coefficients.

Such transformation and encoding schemes and alike may be new to the field of IBR, but they are not to the community as a whole, of course. The coding scheme employed in [61], for instance, has formerly been introduced to the field of volume rendering by the same authors [60].

However, fully 4-dimensional wavelet transforms are exploited by several authors. In [98], for example, a 4D non-standard Haar transform is applied. Since the Haar wavelets are so-called tree wavelets, i.e. the wavelet functions $\psi_{m,k}$ of each level $m$ have disjoint support, all coefficients can easily be re-organized in a coefficient tree. The Wavelet stream implements a suitable format for progressive storage of a linearized stream of such coefficients of arbitrary vector-valued data, such as RGB or YUV. The wavelet stream therewith supports progressive transmission of compressed light field data, too. In addition, interactive rendering speeds are possible due to random access to coefficients while decoding and support by a 3-level caching system.

Comparable work has been presented before in [71] where a fully 4D transform using the Haar basis and following the non-standard decomposition is applied, too. The Haar basis' tree structure is also utilized in a very similar way, constructing a tree-like structure called the Wavelet coefficient tree which basically is a sparse hexa-decary tree with 16 children per node. This format also enables reconstructions at interactive rates and features comparable compression gains, but re-ordering in order to support progressive transmission has not been discussed. Instead, the authors have included detailed discussions about parameterization issues in their setting, the choice of a decomposition strategy for higher dimensional wavelet transforms as well as the choice of wavelet basis functions.

**Computation.** It is observed in [71] that the costs for point-wise reconstructions are logarithmic in the number of light field samples, when the properties of the wavelet coefficient tree structure and the wavelets themselves are utilized. As was reported, rendering at interactive speeds are feasible when taking advantage of this observation, in conjunction with multi-leveled caching mechanisms.
The progressive inverse wavelet synthesis [139] (PWIS) represents a scheme that implements this observation somewhat more consistent. It allows for random access and partial decoding of wavelet transformed and compressed 3-dimensional data, based on inverse lifting operations [34, 124]. Partial decoding means access to and decoding of only the portion of data that is actually necessary to reconstruct a queried view, in order to reduce the computational overhead for reconstructions to a minimum.

Comparable work that also takes advantage of the possibility to only involve small parts of wavelet projected data for an inverse transform has lately been introduced to the field of volume rendering [51, 29, 30], too. However, the idea of so-called local operators, not only for inverse transform operations, will be discussed in depth in Chapter 5. In particular, Section 5.1 will again return to the above mentioned publications.

The hierarchical representation of the light field data, projected to wavelet space using a fully 4-dimensional non-standard transformation, computed as an integral part of the image data acquisition, forms a core topic of this thesis.

The properties of the wavelet transform are exploited in order to construct locally working operators, for both inverse transforms and projection operations (Chapter 5). Secondly, they are utilized to construct a custom data structure supporting storage and modification of compressed, highly dynamic wavelet transformed data (Section 6.2).
In the course of the development of the main contributions of this thesis, a prototype system was built in order to test and evaluate all parts, first individually, then with interaction. This test bed is made up of several conceptual parts which will be introduced in this chapter.

Albeit the prototype system itself is not understood as being one of the primary contributions of this thesis—according to the brief list given in the introductory chapter—it may be instructional to first win an overview of the system before dipping into the details of data representation and the set of operators working on it in the succedent chapters.

There are, in principle, four conceptual components that make up the system as a whole as is depicted in Figure 3.1. These four parts—the acquisition, the parameterization, the core part as well as the viewing component—will be discussed in the following sections individually.

![Figure 3.1 Conceptual overview of the prototype system.](image)

### 3.1 ACQUISITION COMPONENT

The acquisition component's task is to provide previously unused input imagery to the core, with the position and orientation information of the corresponding camera attached. The main reason for this separation of acquisition tasks from the core compo-
ponent is to keep the system's core independent of the source of input. The implementation features two basic alternatives for how to connect an input source to the core, which are the (1) batch mode and the (2) video mode (Figure 3.2). The former mode reads from a readied stack of image files whereas the latter reads from an on-line video source.

When using the batch mode, the acquisition component delivers individual new input images from a bulk of available images to the core on demand. Such sets of images are previously set up in either of the two following ways:

- **Raytracing:**
  An arbitrarily sized set of images of any resolution may be generated using a raytracing program, thus providing synthetic input. The advantages of this solution are obvious. Synthetic images are cheap to generate and always on-hand. Furthermore, the pose information, i.e. position and orientation, of the corresponding camera is readily available and accurate. Hence, no complicated calibration procedure is required. The ALIAS|Wavefront Studio|Tools offer a convenient environment to first arrange or adapt a synthetic scene, which can then be exported using the scene description language (SDL). Files in the SDL format may be modified and straightly used as input to the raytracing component of this software package [ALIAS®Tools].

- **Real imagery:**
  Real images may be captured, adapted if necessary and stored on disk, thus providing real-world input available on demand. Of course, calibration of intrinsic camera parameters becomes inescapable when following this approach. Extrinsic calibration may be imperative, too, in case no other method for an accurate camera pose estimation is available. However, a stack of calibrated real-world images, stored on disk, is the method that has been employed in the course of the evaluation which is documented in detail in Chapter 7.

  It is important to note that images are always readily available to the core if using the batch mode. They are not being generated when needed as is done in [107].
On the other hand, using the video mode, the acquisition component reads image data directly from an on-line camera. This has not been used much nor has it been evaluated in the course of this thesis, since it demands for either an external tracking mechanism or a very time-consuming process based on image understanding in order to get the pose information of the camera at any time. However, the integration of a decent external camera tracking system will demand for a few engineering problems to be solved, e.g. accurately associating pose information measured by the tracking mechanism at a certain rate with pictorial information from the camera shot at a different rate. External tracking may be one among a few directions for future work, though, as will be discussed in Section 8.3 in the last chapter.

3.2 PARAMETERIZATION COMPONENT

The parameterization module is needed to transform image data and its associated camera pose information into samples and their corresponding 4-dimensional indices, as is required by the light field representation. It is important to note that a single sample in 4D consists of again four components, namely the three color channels RGB plus a fourth channel storing a weight. Consequently, it is multi-dimensional and multi-channel data, i.e. 4-dimensional αRGB data, that all algorithms have to deal with.

The implementation offers two different parameterizations to be chosen from. Even though this choice may be done at run-time, the two parameterizations cannot be mixed, of course. That is, light field data generated with one parameterization must be viewed and refined with the same parameterization.

The two options are the (1) cylinder-plane parameterization and the (2) sphere-plane parameterization (Figure 3.3). Both alternatives will be discussed in the following sections and further deepened in Section 6.1.

![Parameterization component](image-url)
Cylinder–plane parameterization. The cylinder–plane parameterization has first been presented in [82]. It parameterizes each viewing ray by way of computing the intersections with basic 3D shapes, i.e. a cylinder and a plane, in particular. The cylinder thereby encloses the 3D scene of interest which is centered at the cylinder's origin. The plane is located inside the cylinder, centered, aligned parallel with its axis and oriented perpendicular to the current viewing direction. Using this configuration, each oriented line may be identified by its four intersection parameters \((p, h)\) on the cylinder and \((s, t)\) on the plane. The basic setting is shown in Figure 3.3, on the left side. A more detailed view will be given in Figure 6.1 in Chapter 6.

The cylinder–plane parameterization has various advantages. Most importantly, it provides full horizontal coverage and thus, it is not necessary to combine multiple parameterization spaces for extended range capturing, as is needed with the standard 2–plane setting (Section 2.2.2 on page 18). Moreover, in contrast to a multi–slab arrangement, the cylinder–plane parameterization inherently avoids artifacts caused by the disparity problem (Section 2.3.2 on page 33) for horizontal movements due to its construction.

On the downside, a difficulty of the cylinder–plane parameterization may arise when moving the camera off the \(z = 0\) plane to an elevated position, while still pointing to the origin. In such situations, foreshortening effects will occur due to the perspective projection of the camera's image plane onto the \(st\)–plane, i.e. the plane inside the cylinder. These effects and possible work–arounds will also be restated and deepened in Section 6.1.

However, the reason for the perspective distortion is the fixed alignment of the plane with the cylinder's axis. Accordingly, it may be completely avoided by softening this restriction. This insight leads to the second parameterization introduced below.

Sphere–plane parameterization. The sphere–plane parameterization may basically be constructed from the cylinder–plane setting by replacing the enclosing 3D shape, i.e. the cylinder, with a sphere. Again, the plane is centered inside the sphere at its origin (Figure 3.3, right side). It meets, however, no other alignment than an orientation perpendicular to the viewing direction, thus avoiding foreshortening completely. The sphere–plane parameterization also avoids artifacts caused by the disparity problem due to its construction, and if sampled uniformly, it does not suffer from the disparity problem at all (Section 2.3.2).

By using the sphere–plane setting, each ray may be identified by its four intersection parameters \((\varphi, \theta)\) on the sphere and again \((s, t)\) on the plane.

The sphere–plane parameterization as was derived in the course of this thesis shares many analogies with the DPP [16]. The major difference, however, is given by the DPP's characteristic to explicitly generate uniformly sampled light field representations.

3.3 SYSTEM'S CORE COMPONENT

The core element contains the two cardinal ingredients of the system as a whole, namely (1) the light field data itself and (2) all operators that generate, update and read from this data. The interaction of all operators and all other system components with the data occurs in two separate modes which will be discussed in the following sections. The matter of mode switching will be briefly addressed afterwards in Section 3.3.3.
3.3 SYSTEM'S CORE COMPONENT

3.3.1 Approximation Mode

During the first mode, the approximation mode, a first light field estimate is computed by projecting an initial set of input images using a given parameterization. A quadra-linear basis function is employed for this projection, similar to the procedure in LUM [45]. The resulting light field is sampled only coarsely and may even contain undefined locations with no meaningful data, especially in cases of small sets of input images.

In order to improve on this first estimate, a hierarchical scattered data interpolation (HSDI) based on the scheme described in [45] is performed. Besides the multi-dimensional interpolation, the strategy pursued during the approximation mode additionally builds up a complete wavelet representation of the data. The result of interpolation and transformation to wavelet space—which is a monolithic, simultaneous procedure—is an extended multiresolution analysis (eMRA) as is indicated in Figure 3.4 below. Its generation and representation will be introduced in Chapter 4.

Note that the viewing component of the system already connects to the first estimate of the light field itself, i.e. rendering is possible at any time, be it from the initially projected image data or be it from the interpolated and wavelet transformed representation.

Further note that if the compression mechanism is set to be active, all mentioned processing steps—initial projection, hierarchical interpolation and transform as well as read-out for rendering—work in the very same way on the custom compressed memory representation that will be presented in detail in Section 6.2.

![Core component - approximation mode](image)

FIGURE 3.4 Close-up of the core component, approximation mode.

3.3.2 Oracle Mode

Once an initial estimate is found and transformed to wavelet space, the core switches to its second mode, the so-called oracle mode. In this mode, aiming at a progressive refinement of the initial estimate, the core continuously grabs previously unused images from the acquisition component that contain new information.

Before such new image samples are further processed, the core first decides on the acceptance of each of them individually, based on an analysis of the difference between the
input image and the corresponding reconstructed view from the current camera position. Accepted image samples are incrementally inserted as parameterized data into the hierarchy, rejected ones are discarded (Figure 3.5). The core thus acts as a kind of an oracle, thereby giving the second mode its name.

Note that the oracle decides on each input image individually. Hence, the oracle does not aim at finding the optimal set of images representing a given 3D scene as a whole, but rather at locally minimizing the amount of data being acquired. As a result, the set of selected image samples will depend on the order in which the images are being presented to the core. The question of how to acquire the optimal set has not been answered yet. Moreover, PS [20] gives a guidance for the minimal sampling rate for anti-aliased light field rendering but does not discuss how to acquire this minimal set either.

In order to minimize the computational cost of an update cycle throughout the oracle mode, so-called local operators are being introduced that work on a clearly defined subset of the light field data instead of the complete 4D data set as a whole. Since one update cycle involves a reconstruction operation for rendering, a comparison and eventually a progressive refinement of the light field data afterwards, a small set of local operators is needed. That is, both a local reconstruction operator for the purpose of rendering and a local projection operator for data insertion that directly alters the eMRA are required. The set of local operators will be addressed in detail in Chapter 5.

As was already reported in [75] a single image may be understood as a 2-dimensional slice through the 4D light field data set. Since both the reconstruction and the insertion operator basically work on images, they may also be understood as operators performing their task on a 2D subspace. In fact, they both operate on small sets of slices through the 4D data. The thus achievable, substantial advantage of slicing regarding complexity and computational efficiency will be addressed in detail in Chapter 5 whereas its impact on the parameterization will be discussed in Section 6.1.3.

As is the case during the approximation mode, all operators needed in the oracle mode also work on the compressed memory representation in the exact same way.

**FIGURE 3.5** Close-up of the core component, oracle mode.
3.3.3 Mode Switching

Given the two presented modes of the core component, the question might arise at which point the core switches from the first to its second mode. Since the system allows the generation of reconstructions, i.e. renderings at any time, namely already during the approximation mode, the decision mechanism used during the continuous refinement of the oracle mode may be borrowed to decide on the switch of modes, too. As is done during the oracle mode new imagery acquired during the first mode may also be compared with its corresponding reconstruction from the already stored data. Based on a threshold criterion applied to an error measure computed from such a difference image the system can decide when to switch from the 1st mode to local refinements performed in the 2nd mode.

Since the light field data acquired during the approximation mode most probably still contains undefined locations before the HSDI has been performed it is difficult to tune such a suitable threshold. As a consequence, the HSDI process and the cMRA construction with the subsequent mode switch is triggered by the user in most cases.

3.4 VIEWING COMPONENT

The viewer module provides all necessary functions concerning display of input and output of the system. Additionally, all system control mechanism needed to use and guide the system are also part of it.

The different display elements are the main light field viewer plus three special purpose viewers for 1) the current input image, 2) the reconstructed image at the corresponding viewpoint and 3) a 3D visualization of the spatial camera position and orientation of accepted and rejected input images in relation to the object’s position (Figure 3.6). The latter viewer may be understood as some sort of user guidance system by visualizing more densely and still sparsely sampled locations around the object of interest. Both the main light field viewer as well as the 3D visualization support interactive free viewpoint selection based on quaternion rotation [105].

**FIGURE 3.6** Close-up of the viewing component.
Seite Leer / Blank leaf
The data representation is the central part of the acquisition and processing system described in this thesis—in combination with the set of operators interacting with it, of course. This chapter's focus lies on the initial construction and all subsequent modifications of the representation. The physical storage issue is one of the two central topics of Chapter 6 and will thus not be further discussed in this chapter. In particular, Section 6.2 presents the custom-tailored data structure that is used for the compressed data representation.

In the following sections a short compendium of the requirements for the representation will be given first before moving on to the basic concepts and brief introductory surveys of function approximation and wavelet theory (Sections 4.2 & 4.3), hierarchical concepts that may be found in the field of IBMR (Sections 4.4 & 4.5) and finally the construction and handling of the representation itself (Sections 4.6–4.9).

4.1 NECESSITIES

Since there is quite a number of requirements an appropriate representation has to satisfy, it may be well important to first understand these necessities. The following list gives a commented overview of these needs.

- Handling of incomplete sampling:
  The core component of the system builds a first estimate of the 4D light field function during the approximation mode, based on an initial set of input images. Depending on the size of this set and depending on the distribution of the individual images it may easily happen that the light field function is only partially defined after the initial projection of this input imagery. That is, parts of this function may remain entirely undefined. Or in other words, a suitable data representation has to be able to (1) handle an incompletely sampled function and has to (2) feature suitable approximation power.
Handling of **adaptive sampling:**
The light field function may not only be incompletely sampled, but there may be regions with unequal densities of samples, too. Hence, a representation needs to manage adaptive sampling densities.

These first two requirements make the incorporation of a scattered data interpolation mechanism into the representation inevitable. Both effects, however, incomplete sampling and adaptive densities of samples do occur due to the random selection of input images.

**Free choice of interpolation filter kernel:**
Regardless of what a chosen representation looks like and in which way it is being encoded, it is highly desirable to keep it independent of a particular filter kernel employed since the approximation of the light field function is significantly influenced by the quality and the characteristics of the interpolation filter.

**Support for progressive & incremental operations:**
During its second mode, the so-called oracle mode, the core aims at progressively refine the initial estimate of the light field function. Since invalidating and re-generation of the complete 4D representation in case of an update or any other mutation is far too costly and hence not a very smart strategy to follow any support for incremental updates is imperative and must therefore be allowed by a representation.

**Support for fast interaction operators:**
Having the possibility to incrementally update a representation, the operators performing such an update need to be as efficient as possible. Consequently, support for incremental updates only is not sufficient, but the operator that performs a data insertion or modification needs to work fast, too. Of course, the same demand holds for an operator reading from the representation, i.e. reconstructing a view. Since both of these operators basically work on 2-dimensional slices out of the 4-dimensional data, i.e. on images, they are expected to work much more efficient than an operation performed on the 4D representation as a whole. To be more technical, they are expected to perform in the order of $O(N^2)$ instead of $O(N^4)$, with $N$ denoting the size of any dimension of some data set.

**Support for compression:**
Because methods in the field of IBMR usually demand for vast amounts of storage for their primary data, compression is mandatory. Thus, support for a progressive and efficient compression scheme with high compression gain is very important, thereby still allowing the operators mentioned before to do their work efficiently.

In order to meet all of these requirements, a so-called **extended multiresolution analysis** (eMRA) was designed as a unified representation for sampling, interpolation, compression and rendering. A fully 4-dimensional wavelet transform is being applied, with the additional freedom of an independent interpolation filter kernel.

However, before moving on to wavelet theory and the notion of multiresolution analysis, a short summary of approximation theory, i.e. the results needed in the context of this thesis, will be given first.
4.2 APPROXIMATION OF FUNCTIONS

Generally speaking, approximation theory is the mathematical study of how given quantities can be approximated by others—usually simpler ones—under certain conditions. Or, speaking of functions, it is the study of how to approximate continuous functions by other functions that depend only on a finite number of parameters. Of course, approximation theory also studies the size and properties of the error introduced by an approximation. However, approximations are often obtained by power series expansion in which the higher order terms are dropped at some point.

Approximation theory as a whole is a vast field that clearly goes beyond the scope of this thesis. Nevertheless, compact introductory textbooks are available, e.g. [1] approaches approximation theory from the viewpoint of functional analysis, whereas [104] provides a concise introduction to the most significant methods in the field with particular emphasis on approximation by polynomials. Besides, an additional historical background may be found in [24] while the focus of [25] is put on new developments in approximation theory that have come up over the last two decades, including wavelet theory.

The following paragraphs will just briefly summarize some of the outcomes of the theory of function approximation in the continuous case that are needed later on. In particular, the focus is put on finite dimensional approximations of functions in \( L^2(\mathbb{R}) \) by other basis functions, following Gauss’ ideas of minimizing a squared distance measure.

The space \( L^2(\mathbb{R}) \) denotes the space of square–integrable functions. It is defined as the space of Lebesgue–measurable [11] functions \( f \) which have to satisfy the finite energy condition in Equation 4.1 below. Concise definitions of relevant spaces and norms may be found in [11], for example.

\[
\|f\|_2^2 = \int_{-\infty}^{+\infty} |f(x)|^2 \, dx < \infty \quad (4.1)
\]

A given function \( f(x) \) may be well approximated by another function \( g(x) \) over the interval \([a,b]\) if the squared distance between these two functions is being minimized, as is expressed in Equation 4.2 below, where \( w(x) \) denotes an additional weighting function with \( w(x) > 0 \) in \([a,b]\).

\[
F = \int_a^b w(x) \cdot (f(x) - g(x))^2 \, dx = \min \quad (4.2)
\]

A good choice for the approximation function \( g(x) \) is a weighted sum of a finite set of linearly independent functions \( g_0(x), g_1(x), \ldots, g_n(x) \), i.e.

\[
g(x) = \sum_{i=0}^{n} c_i \cdot g_i(x) \quad (4.3)
\]

Using this approach, the coefficients \( c_i \) must be determined such that the functional \( F \) gets minimized. Thus, the following conditions must be satisfied:

\[
\frac{\partial}{\partial c_i} F = 0 \quad i = 0, 1, \ldots, n \quad (4.4)
\]
The resulting system of normal equations shown in Expressions 4.5 has a unique solution since the set of functions \{g_i(x)\} has been assumed to be linearly independent.

\[
\sum_{i=0}^{n} c_i \cdot \langle g_i \mid g_j \rangle = \langle f \mid g_j \rangle \quad \text{with} \quad \langle g_i \mid g_j \rangle = \int_a^b w(x)g_i(x)g_j(x)dx
\]

The coefficients \(c_i\), however, can be computed without solving the complete system of equations in case the functions \{g_i(x)\} chosen as basis functions are orthogonal, i.e. in case the inner product of any two functions is equal to zero:

\[
\langle g_i \mid g_j \rangle = 0 \quad i \neq j \tag{4.6}
\]

Moreover, the basis functions are orthonormal if \(\langle g_i \mid g_j \rangle = \delta_{ij}\) holds, with \(\delta_{ij}\) being the Kronecker delta function. Thus, the coefficients \(c_i\) can be computed directly from the normal equations in Expressions 4.5 as follows.

\[
c_i = \langle f \mid g_i \rangle \tag{4.7}
\]

In the case of orthogonal basis functions an additional normalization factor must be taken into account. However, due to Equation 4.7, it is in fact quite convenient to pick out an orthogonal basis to work with. Fortunately, sets of linearly independent functions may be made orthogonal using procedures like Gram–Schmidt and others [11]. Table 4.1 below shows a set of orthogonal polynomials along with their weighting function and the valid interval that may be obtained starting from the power functions \(g_i(x) = x^i\) with \(i = 0, 1, \ldots, n\).

<table>
<thead>
<tr>
<th>Orthogonal polynomial</th>
<th>Weighting function</th>
<th>Interval in effect ([a,b])</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legendre (P_n(x))</td>
<td>(w(x) = 1)</td>
<td>([-1,1])</td>
</tr>
<tr>
<td>Tschebyscheff (T_n(x))</td>
<td>(w(x) = 1/\sqrt{1-x^2})</td>
<td>([-1,1])</td>
</tr>
<tr>
<td>Laguerre (L_n(x))</td>
<td>(w(x) = e^{-x})</td>
<td>([0,\infty))</td>
</tr>
<tr>
<td>Hermite (H_n(x))</td>
<td>(w(x) = e^{-x^2/2})</td>
<td>((-\infty,\infty))</td>
</tr>
</tbody>
</table>

Although the above bases account for many applications in practice, they are by far not suitable for any problem. To give an example, periodic functions, e.g. \(f(x + 2\pi) = f(x)\) with \(x \in \mathbb{R}\) are best approximated using finite Fourier series which, in turn, use an orthogonal basis of trigonometric functions. However, note that any finite interval \([a,b]\) may easily be transformed to \([-1,1]\), for which many bases are defined.

In case of a non–orthogonal basis, the coefficients \(c_i\) cannot be calculated according to Equation 4.7, i.e. \(c_i \neq \langle f \mid g_i \rangle\). Fortunately, one of the fundamental results of linear algebra states that for any finite–dimensional primal basis \{\(g_i(x)\)\} there always exists a so–called dual basis \{\(\widetilde{g}_i(x)\)\} such that Equation 4.8 holds.
If the above relation is to hold for every function $f$ spanned by $\{g_j(x)\}$, then it follows that $\langle g_i \mid \tilde{g}_j \rangle = \delta_{ij}$, as is shown in Expressions 4.9.

\[
c_i = \langle f \mid \tilde{g}_i \rangle = \int_{-\infty}^{+\infty} f(x)\tilde{g}_i(x)dx = \int_{-\infty}^{+\infty} c_j g_j(x)\tilde{g}_i(x)dx \Rightarrow \langle g_j \mid \tilde{g}_i \rangle = \delta_{ji} \tag{4.9}
\]

It is important to note that if the set of functions $\{g_j(x)\}$ is orthonormal, then the basis is said to be self-dual, i.e. primal and dual bases are the same: $g_j(x) = \tilde{g}_j(x)$. Thus, the inner product matrix is equal to the identity matrix:

\[
\langle g_j \mid g_i \rangle = \delta_{ji} \iff \begin{bmatrix} \langle g_j \mid g_i \rangle \end{bmatrix} = I \tag{4.10}
\]

In case of an orthogonal basis, the inner product matrix is not the identity anymore, of course, but some diagonal matrix. Moreover, the inner product matrix of a basis that is nearly but not perfectly orthogonal shows a band structure.

An important question that remains is how the error introduced by a truncated approximation series can be controlled. Supposed a function $f(x)$ is being perfectly expanded by an orthonormal basis $\{g_j(x)\}$, then Equation 4.11 holds.

\[
f(x) = \sum_{i = 0}^{n} c_i g_i(x) \tag{4.11}
\]

Given an integer $N$ with $1 \leq N < n$, the question is how to find the best approximation of $f(x)$ using the truncated series with just $N$ basis functions. Thanks to the properties of orthogonality the $L^2$-error introduced may be computed as follows:

\[
\|f(x) - f_{\text{approx}}(x)\|_2^2 = \left\| \sum_{i = N + 1}^{n} c_i g_i(x) \right\|_2^2 = \langle \sum_{i = N + 1}^{n} c_i g_i(x) \mid \sum_{i = N + 1}^{n} c_i g_i(x) \rangle = \sum_{i = N + 1}^{n} |c_i|^2 \tag{4.12}
\]

Thus, the magnitude of the coefficients $c_i$ fits exactly to the fraction of energy provided by the corresponding basis function $g_i(x)$. According to Equation 4.12, however, a globally optimal approximation $f_{\text{approx}}(x)$ for a given integer $N$ can be achieved by sorting the coefficients $c_i$ by their magnitude and subsequently rejecting the $N$ smallest ones. Using a fast sorting algorithm, the subset of coefficients to be dismissed may be computed in $O(n \cdot \log n)$ [68].
4.3 WAVELETS

The following subsections briefly review various important aspects of wavelet theory. Again, they are not meant to be understood as a complete introductory compendium nor do they substitute any textbook about wavelets and their applications. They rather review a set of aspects that are important in the connexion of this thesis.

More detailed but still concise introductions to wavelet theory, however, may be found easily, e.g. [79, 118]. On the other hand, textbooks such as [27, 33, 88, 119], which are all quite renowned in the meantime, offer comprehensive introductions to wavelets and their various applications.

4.3.1 Overview

The wavelet theory provides a sound mathematical framework for the construction of multiresolution representations of 1-dimensional or even multi-dimensional functions.

The fundamental mechanism to generate such a representation, however, is impressively simple: Starting with a 1D function, its wavelet representation is computed through projection onto two sets of basis functions, which are (1) the scalar and (2) the wavelet bases. The scalar basis generates a set of approximations of the input function in variable resolutions. The wavelet basis on the other hand extracts the fine detail information that is needed to completely reconstruct the original input function from the set of approximations. Such a reconstruction from a wavelet representation requires the dual bases [27] of the scalar and the wavelet bases. This technique has been extended to 2D and also higher dimensional functions using the tensor product concept (Section 4.3.5).

The wavelet transform and the fast wavelet transform (FWT) [27, 33, 87] used for discrete function data in particular are very scalable and highly efficient tools. Firstly, the time complexity of the analysis and synthesis filters which are needed to construct and evaluate a wavelet representation of some function is linear in the size of the input data. Secondly, the size of the hierarchical structure of a wavelet representation is the same as the size of the input data.

The hierarchical structure—also referred to as the wavelet pyramid—may be packed into special structures such as a Zerotree [113] or SPIHT [106]. They provide even more compact representations due to compression techniques with clever requantization schemes.

Due to their flexibility and efficiency wavelets have been successfully introduced to many different fields. As an example, wavelets are well used to design the next generation image compression algorithms [26, 116].

An important characteristic of the wavelet theory and its mathematical framework that is certainly one of the reasons for the broad success is the precise error control provided. Exact error bounds of a given wavelet–based approximation may be computed—depending on the orthogonality of the scalar and the wavelet bases. This feature is being used in many applications, e.g. geometric compression [49] or volume rendering [80] where error–bounded approximations of large models are computed.

The smoothness of such approximations, however, is dependent on the smoothness of the basis function applied. As a consequence, the basis' smoothness properties are an important argument when choosing a certain basis for the representation of a given function (Section 4.3.3).
As a matter of course, the wavelet theory has its limitations, too. Most importantly, it expects the samples of a function to be aligned on a regular grid. The introduction of the second generation wavelets, however, has eased this restriction greatly. This new framework does not rely on the construction of the scalar and the wavelet basis functions by using the 2-scale relation \[27\]. Instead, input data is transformed following the lifting scheme \[125\] which stands for a more flexible tool, enabling wavelet representations over more general domains. However, it is the classical first generation wavelet theory that is made use of in the context of this thesis.

4.3.2 Multiresolution Analysis (MRA)

One possible way to introduce wavelets is through multiresolution analysis (MRA), i.e. by defining a multiresolution analysis. The alternative, through the continuous wavelet transform, will not be further pursued at this point. However, a more thorough introduction to the concepts of multiresolution analysis may be found in \[120\] for example. The notion of MRA has primarily been introduced by Stéphane Mallat in 1989 \[87\].

A multiresolution analysis of \(L^2(\mathbb{R})\) is defined as a sequence of closed subspaces \(V_m\) of \(L^2(\mathbb{R})\) with \(m \in \mathbb{Z}\) and the following properties:

1) Nestedness, i.e. nested sequence of subspaces, from coarse to fine:

\[\{0\} \subset \ldots \subset V_{m+1} \subset V_m \subset \ldots \subset V_1 \subset V_0 \subset \ldots \subset L^2(\mathbb{R}) \quad (4.13)\]

2) Scale invariance:

\[f(x) \in V_m \Leftrightarrow f(2^m \cdot x) \in V_0 \quad (4.14)\]

3) Shift invariance:

\[f(x) \in V_0 \Rightarrow f(x-k) \in V_0 \quad k \in \mathbb{Z} \quad (4.15)\]

4) Completeness: The union of all \(\{V_m\}_{m \in \mathbb{Z}}\) spans \(L^2\).

\[\bigcup_{-\infty}^{+\infty} V_m \text{ is dense in } L^2(\mathbb{R}) \text{ and } \bigcap_{-\infty}^{+\infty} V_m = \{0\} \quad (4.16)\]

5) Riesz basis \[11\]: Any \(f \in V_m\) has a unique representation as a linear combination of some functions \(\phi_{m,k}(x)\), i.e.

\[\exists \phi(x) \in V_0 \text{ such that } \{\phi(x-k) \mid k \in \mathbb{Z}\} \text{ forms a Riesz basis of } V_0 . \quad (4.17)\]

These properties lead to the following observations. Firstly, since \(\phi(x) \in V_0\) and \(V_1 \subset V_0\), a sequence \((h_k) \in L^1(\mathbb{Z})\) exists such that the scaling function \(\phi(x)\) satisfies

\[\phi(x) = 2 \cdot \sum_{k} h_k \cdot \phi(2x-k) . \quad (4.18)\]

The above Equation 4.18 is the well-known 2-scale relation which also goes by several different names, such as refinement equation, dilation equation or the 2-scale difference equation. The sequence \((h_k)_{k \in \mathbb{Z}}\), however, is required for the implementation of the FWT and is heavily used by the algorithms described in Chapter 5.
Moreover, a set of functions \( \{ \phi(x - k) \}_{k \in \mathbb{Z}} \) spans the subspace \( V_0 \). Likewise, the set of functions \( \{ \phi_{mk} \} \) with

\[
\phi_{mk}(x) = 2^{-\frac{m}{2}} \cdot \phi(2^{-m}x - k)
\]  

(4.19)

constitutes a Riesz basis for the subspace \( V_m \). Hence, each function \( f \in V_m \) can be expressed as a linear combination, i.e. a weighted sum of scaling functions \( \phi_{mk} \) of approximation level \( m \) and some corresponding coefficients \( c_{mk} \):

\[
f(x) = \sum_k c_{mk} \cdot \phi_{mk}(x)
\]

(4.20)

It is important to note that \( \phi(x) \) is a valid scaling function of \( L^2(\mathbb{R}) \) if and only if it also forms a partition of unity with its integer translates, as is expressed in Equation 4.21 below.

\[
\sum_k \phi(x - k) = 1
\]

(4.21)

The question remains how any detail information of a subspace \( V_m \) can be secured and how the set of subspaces can be combined and complemented with the detail information in order to form a hierarchical representation of a given function. To this end, a space \( W_m \) is introduced that complements \( V_m \) in \( V_m \), i.e.

\[
V_m = V_m \oplus W_m
\]

(4.22)

In other words, the space \( W_m \) contains the detail information that is needed to move from an approximation at resolution level \( m \) to an approximation at \( m - 1 \).

A function \( \psi \) is defined as being a wavelet if the collection of functions given as \( \{ \psi(x - k) \mid k \in \mathbb{Z} \} \) forms a Riesz basis of \( W_0 \). The collection of wavelet functions \( \{ \psi_{mk} \mid m, k \in \mathbb{Z} \} \) is then a Riesz basis of \( L^2(\mathbb{R}) \). As is expressed below, the definition of \( \psi_{mk} \) is very similar to the definition of the scaling function that was given in Equation 4.19.

\[
\psi_{mk}(x) = 2^{-\frac{m}{2}} \cdot \psi(2^{-m}x - k)
\]

(4.23)

Thereupon, the subspaces \( \{ W_m \}_{m \in \mathbb{Z}} \) are spanned by wavelet functions \( \psi_{mk} \) in a very similar way as compared to the spaces \( \{ V_m \}_{m \in \mathbb{Z}} \) that are spanned by scaling functions \( \phi_{mk} \). Again, the wavelet functions \( \psi_{mk} \) are constructed from a mother function by binary dilation and dyadic translations, as is expressed in Equation 4.23 above.

Since the wavelet \( \psi_{mk} \) is an element of \( W_m \) and there of \( V_{m - 1} \), too, a sequence \( (g_k) \in l^2(\mathbb{Z}) \) exists such that

\[
\psi(x) = 2 \cdot \sum_k g_k \cdot \phi(2x - k)
\]

(4.24)

The above Equation 4.24 is the 2–scale relation for the wavelet bases.
Further note that the other properties mentioned for the scaling function $\phi(x)$ also hold for the wavelet function $\psi(x)$.

The introduction of the hierarchical set of basis functions allows a hierarchical representation of some function $f(x)$. More precisely, it is now possible to decompose some function $f(x) \in V_{m_0}$ into a sequence of projections into the subspace $V_M$ and the spaces $\{W_m\}_{m_0 < m \leq M}$. In other words, the function $f(x) \in V_{m_0}$ can be written as a linear combination of a coarse approximation at level $M > m_0$ and additional detail information at levels $m$ with $m_0 < m \leq M$, as is expressed in Equation 4.25 below.

$$f(x) = \sum_k c_{mk} \phi_{Mk}(x) + \sum_{m = m_0 + 1}^M \sum_k d_{mk} \psi_{mk}(x)$$ (4.25)

The coefficients $c_{mk}$ and $d_{mk}$ are determined by the inner products of $f(x)$ with the corresponding basis function, i.e.

$$c_{mk} = \langle f | \phi_{mk} \rangle \quad d_{mk} = \langle f | \psi_{mk} \rangle$$ (4.26)

Such hierarchical representations support the construction of multiresolution editor concepts, i.e. highly efficient editors that operate on the different levels of approximation of a wavelet pyramid. To give an example, multiresolution image editors that operate in 2D are described in [119].

### 4.3.3 Properties of Wavelets

Many different types of wavelet basis functions are available in the meantime. Some of the best known examples certainly include the Haar basis, the Daubechies and Gabor families as well as the endpoint-interpolating B-splines. There are many more, such as Shannon, Meyer or Battle-Lemarié wavelets. Hints to further readings may be found in all previously given references concerning wavelet theory and applications.

As a matter of course, every basis features different properties which makes one or the other more suited to be used in a given situation. Some of the most important properties will be reviewed in the following sections. A more detailed discussion may be found in [64] for example.

- **Compact support:**
  Functions that are said to be compactly supported are non-zero over a bounded, relatively small interval only. A more compact support of scaling functions and wavelets gives two important advantages. Firstly, it may greatly improve the efficiency of decomposition and reconstruction operations since fewer functions have to be taken into account for one data location. Secondly, wavelets with small support avoid smearing effects of local detail across bigger areas of the data.
  However, the support of a function $g(x)$ is formally defined as the size of the interval over which the function is non-zero:

$$\text{supp}(g) = \{ x \in \mathbb{R} \mid g(x) \neq 0 \}$$ (4.27)

- **Smoothness:**
  Smooth functions are, in general, best represented with a smooth basis since the smoothness of a wavelet representation depends on the smoothness of the underlying basis function. In case a function shows regions with smooth and other areas with
sharp features, a single wavelet basis will fail to capture all information. The concept of multi-wavelets \([41, 72]\) offers a solution in such situations, but only if the different areas, i.e. smooth regions and areas with sharp features, can be identified reliably. Furthermore, the smoothness of a wavelet basis applied may have impact on applications including compression. As was mentioned before (Section 2.3.3 on page 37) wavelet-based compression is done by disregarding small \(d_{mk}\) and thereby omitting the corresponding basis function (Equation 4.25). Smoother bases usually give better results when functions are approximated by such truncated expressions.

The smoothness of a function, however, may be measured through analyzing the number of continuous derivatives, for example. Finally, it is important to note that greater smoothness mostly comes at the expense of wider function support—and vice versa.

\* Localization:

Wavelets are well localized in both the frequency and the spatial domain—in contrast to many other bases such as the trigonometric basis functions of the Fourier transform which are well localized in the spectral domain only. The relation between the localization of a function \(f(x)\) in the spatial domain, denoted \(\Delta_x^f\), and the localization of its Fourier transform \(F(\omega)\) in the frequency domain denoted \(\Delta_\omega^f\) is defined through the Heisenberg window (Expression 4.28) which is based on Heisenberg's uncertainty principle. Due to this relation, there will always be a trade-off to be followed between good localization properties in the spatial and in the frequency domain. A notable exception is the Gabor function, a modulated Gaussian with optimal localization in both space and frequency because it equals the lower bound given in Expression 4.28 \([131]\). Likewise, B-spline functions asymptotically get at this boundary with increasing order.

Equations 4.29 and 4.30 give the definitions of the spatial spread and the spread in frequency, i.e. the bandwidth, of a function \(f(x)\). Equation 4.29 nicely shows that the size of the interval of compactly supported functions plays a significant role for its spatial localization properties.

\[
\Delta_x^f \cdot \Delta_\omega^f \geq \frac{1}{4\pi} \tag{4.28}
\]

\[
\Delta_x^f = \sqrt{\frac{\int x^2 |f(x)|^2 \, dx}{\int |f(x)|^2 \, dx}} \tag{4.29}
\]

\[
\Delta_\omega^f = \sqrt{\frac{\int \omega^2 |F(\omega)|^2 \, d\omega}{\int |F(\omega)|^2 \, d\omega}} \tag{4.30}
\]

\* Orthogonality:

The attribute of orthogonality applies to bases but to an MRA as a whole, too. The conditions for an MRA to be orthogonal will be stated in Section 4.3.4 along with the more general formulations of semi- and bi-orthogonality. However, it is often very convenient to work with an orthogonal MRA because the \(L^2\)-norm of a function represented by the MRA is directly linked to the norm of the wavelet and scaling coefficients (Sections 4.2 & 4.3.6).
Vanishing moments:
A wavelet function \( \psi(x) \) is said to have \( n \) vanishing moments if the inner product given in Equation 4.31, which defines the \( a \)th moment of a function, equals zero for all \( a < n \). If this is the case, then \( \psi(x) \) is orthogonal to any polynomial of degree \( n - 1 \). Consequently, if some function \( f(x) \) may be well approximated using a polynomial of degree \( a < n \), then the resulting wavelet coefficients \( d \) corresponding to \( \psi(x) \) will be close to zero. If the function \( f(x) \) is a polynomial of degree \( a < n \), the wavelet coefficients will be exactly zero. This, in turn, means that the function's energy is concentrated in a relatively small number of coefficients, mainly scaling coefficients (only scaling coefficients in the latter case), which is an important prerequisite for an efficient compression. In other words, the number of vanishing moments is a meaningful indicator for a given wavelet basis' compression capacities.

It may be interesting to note that the number of vanishing moments must at least be equal one for any wavelet. However, a detailed discussion about vanishing moments of wavelet functions may be found in [88] for example.

\[
\langle \psi(x) | x^a \rangle = \int x^a \cdot \psi(x) \, dx = 0 \quad 0 \leq a < n \quad (4.31)
\]

Symmetry:
For some applications it may be important to use scaling functions and wavelets that are symmetric (or anti-symmetric) about their centers. If they are, the filters have generalized linear phase, which is substantial in signal processing applications. However, it is worth noticeable that, besides the Haar basis, there are no other wavelet bases which are 1) orthogonal, 2) compactly supported and 3) symmetric at the same time [33].

4.3.4 Orthogonality
The following section reviews the conditions for an MRA to be orthogonal. The more general formulations of semi— and bi—orthogonality will be briefly discussed thereafter.

Orthogonal Wavelets. An orthogonal MRA basis is one in which the scaling functions are orthogonal to each other, the wavelets are orthogonal to each other, and each of the wavelet function is orthogonal to every coarser scaling function as is stated in Expressions 4.32. Wavelets satisfying all of these conditions are called orthogonal wavelets.

\[
\begin{align*}
\langle \phi_{mk} | \phi_{mk} \rangle &= \delta_{kk'} \quad k, k' \in \mathbb{Z} \\
\langle \psi_{mk} | \psi_{mk} \rangle &= \delta_{kk'} \quad m \in \mathbb{Z}, m \leq M \\
\langle \psi_{mk} | \phi_{mk} \rangle &= 0
\end{align*}
\]  

(4.32)

Actually, the above conditions define an orthonormal basis by using the Kronecker notation. Note that, in general, an orthogonal basis does not need to have this normalization property.

As an alternative, the last requirement of Expressions 4.32 may also be expressed as \( W_0 \perp V_0 \) for \( m = 0 \). This is a sufficient condition for an MRA to be orthogonal, since the other conditions follow from the properties of an MRA. It follows from Equation 4.22 that the wavelet space \( W_m \) is the orthogonal complement of \( V_m \) in \( V_{m-1} \) and thus, the spaces \( \{ W_m \}_{m \in \mathbb{Z}} \) are all mutually orthogonal. Consequently, the collection of wavelets \( \{ \psi_{mk} \}_{m, k \in \mathbb{Z}} \) forms an orthonormal basis of \( L^2(\mathbb{R}) \).
For the sake of completeness, a scaling function $\phi$ is orthogonal if the set of functions $\{\phi(x - k) | k \in \mathbb{Z}\}$ is an orthonormal basis of $V_0$, as is expressed in the first equation of 4.32 using the Kronecker delta function. Because of Equation 4.19, the collection of functions $\{\phi_{mk}\}_{k \in \mathbb{Z}}$ therewith forms an orthonormal basis of $V_m$.

Likewise, an orthogonal wavelet $\psi$ is a function such that the set of functions $\{\psi(x - k) | k \in \mathbb{Z}\}$ is an orthonormal basis of $W_0$ and $\{\psi_{mk}\}_{k \in \mathbb{Z}}$ makes up an orthonormal basis of $W_m$.

Widely used orthonormal wavelet bases are—besides the well-known Haar basis—the Daubechies wavelet families [33]. Both the wavelet and the scaling functions are compactly supported. For a given number of vanishing moments $n \in \mathbb{N}$, Daubechies wavelets feature a minimum sized support of $2n - 1$ [88]. However, they are not symmetric and an analytic representation does not exist for these functions.

**Semi-orthogonal wavelets.** The limitation to orthogonal constructions of wavelets may be overly restrictive. In fact, if smooth and symmetric wavelets with compact support are desired, orthogonality must be sacrificed—remember that the Haar basis is the only symmetric, compactly supported and orthogonal wavelet basis. If the requirement for orthogonal basis functions is dropped, compactly supported and symmetric (or anti-symmetric) wavelets can be constructed [28, 132]. Such wavelets are at least orthogonal to all coarser scaling functions on a given level of resolution, though not necessarily orthogonal to each other, but still generating an orthogonal MRA. Thus, compared to Expressions 4.32, it is only the last condition that applies:

$$\langle \psi_{mk} | \phi_{mk} \rangle = 0 \quad k, k' \in \mathbb{Z} \quad m \in \mathbb{Z}, m \leq M$$

(4.33)

Wavelets designed to meet this less restrictive criterion are referred to as semi-orthogonal wavelets, or pre-wavelets. Note that orthogonal wavelets are a special case of semi-orthogonal wavelets.

In the case of semi-orthogonality, a dual scaling function $\tilde{\phi}$ and a dual wavelet $\tilde{\psi}$ exist, additionally to the functions $\phi$ and $\psi$. These duals generate a dual MRA with subspaces $V_m$ and $W_m$ that satisfy the following conditions:

$$V_m \perp W_m \quad \text{and consequently,} \quad W_m \perp \tilde{W}_{m'} \quad \text{for} \quad m \neq m'$$

(4.34)

Furthermore, the mutual orthogonality property of the subspaces $W_m$ implies that $W_m \perp W_{m'}$ for $m \neq m'$ and therefore, $W_m = \tilde{W}_m$. This further implies that $V_m = \tilde{V}_m$.

In other words, the sets $\{\phi_{mk}\}$ and $\{\tilde{\phi}_{mk}\}$ both span the same space $V_m$, whereas $W_m$, on the other hand, is spanned by $\{\psi_{mk}\}$ and $\{\tilde{\psi}_{mk}\}$. Thus, the primary and the dual bases generate the same MRA which is in fact orthogonal.

Note that semi-orthogonality does not necessarily imply dual basis functions to be compactly supported. However, an important and well-known example of pre-wavelets is the family of B-spline wavelets [27, 119]. Further constructions of semi-orthogonal wavelets may be found in [64, 90].

**Bi-orthogonal wavelets.** If the requirement for an MRA to be orthogonal is dropped too, the most general form of an MRA is obtained, generated by bi-orthogonal wavelets. As a consequence, Equation 4.33 does not apply anymore.
As is the case for semi-orthogonal wavelets, additional dual scaling and wavelet bases exist that generate a dual MRA. Hence, Expressions 4.34 still hold, which may be stated in a different form as follows:

$$\langle \phi_{mk} | \phi_{mk'} \rangle = \delta_{kk'} \quad k, k' \in \mathbb{Z}$$

$$\langle \psi_{mk} | \psi_{mk'} \rangle = \delta_{kk'} \quad m \in \mathbb{Z}, m \leq M$$ (4.35)

The introduction of a dual MRA—that may be orthogonal or not—offers more flexibility in constructing wavelet bases. More details about the construction of bi-orthogonal wavelets, however, may be found in [27, 31] for example.

### 4.3.5 Multidimensional Bases

All considerations done so far have been restricted to a 1-dimensional setting only. Of course, many applications demand for higher dimensional bases, e.g. wavelet-based image editors and compression mechanisms operate in 2D [116] and wavelet-based volume representations [79] have to deal with 3D bases. Additionally, processing of light field data as is described in this thesis needs to be done in a 4-dimensional setting.

In principle, the extension of 1D bases to a multi-dimensional setting can be constructed in either of two different ways [121]. Firstly, higher dimensional bases may be set up as products of 1-dimensional, separable bases. This approach called tensor product encodes n-dimensional data by applying a 1D transform along each of the major axis successively. While this approach works fine for 1st generation wavelets, it is more complex for 2nd generation wavelets defined over semi-regular domains.

The second possibility applies to non-separable bases such as Gabor functions, for example, that require genuinely multi-dimensional bases. However, such constructions will not be used in the connexion of this thesis and hence, 4D bases are built using the tensor product approach for separable bases.

A natural basis for a bi-variate space $V^2_m = V_m \times V_m$ if a given set $\{\phi_{mk}\}_{k \in \mathbb{Z}}$ forms a basis for $V_m$ is the separable tensor product basis shown below:

$$\phi_{mk,k_2}(x, y) = \phi_{mk_1}(x) \cdot \phi_{mk_2}(y)$$ (4.36)

The question of how to construct the corresponding bases for the remaining complementary spaces $W^2_m$ can be answered in two different ways, either following the standard or the non-standard approach.

**Standard decomposition.** Following the standard decomposition scheme, a given function $f(x, y) \in L^2(\mathbb{R}^2)$, e.g. an image, is transformed by first applying the 1D transform to all rows of the data. In a second step, the 1D transform gets applied to each column of the result from the first transformation procedure. As a result, the 2-dimensional function is translated in a set of detail coefficients plus one single overall average value, which is the only scaling coefficient. Figure 4.1 illustrates the standard decomposition procedure in 2D for two levels of approximation.

A severe disadvantage is that the basis functions used in the standard procedure are combinations of wavelet functions from different resolution levels which prohibits a direct overall control of the level of approximation. Mathematical formulations of these 2D basis functions may be found in [118] for instance. However, the non-standard decomposition overcomes this problem.
Non-standard decomposition. Instead of applying a complete 1-dimensional transform twice, i.e. for all rows first and for all columns afterwards, or vice versa, it is possible to perform a row and a column transformation step at each level, in an alternating manner. This scheme is illustrated in Figure 4.2 below, again in 2D for two levels of resolution. An apparent illustration of a non-standard and a standard decomposition of the same image may additionally be found in [119], Chapter 3.

Contrary to the standard scheme, the non-standard decomposition uses no combinations of basis functions from different levels, therethrough giving good control of the level of approximation.

The non-standard scheme features two more advantages as compared to the standard decomposition. Firstly, it is more efficient in terms of computational complexity as is nicely shown and analyzed in detail in [71]. Secondly, all of the non-standard basis functions have a square support in the 2D case, whereas some of the standard basis functions have also non-square support areas. However, regular support areas, i.e. squared in the 2D case, are an important property for the design of the local operators as is discussed in Section 5.1.

Due to these reasons, the transformation algorithms employed in this thesis all follow the non-standard decomposition scheme.

As a matter of course, the 2-dimensional case discussed in this section may be extended to three and even more dimensions straightforwardly. A detailed discussion including mathematical formulations of the basis functions for the 3-dimensional case may be found in [79].

4.3.6 Coefficient Thresholding

As was briefly brought up in Section 2.3.3, transform compression methods firstly rely on the capabilities of a transformation to concentrate a signal’s or function’s energy in as few coefficients as possible. In a second step, these coefficients are being analyzed, aiming at finding a small set that approximates the original function well enough.

If a wavelet basis with an adequately high number of vanishing moments is used to perform a wavelet transform, quite a few coefficients will vanish accordingly—if the input data is smooth enough (Section 4.3.3). Hence, due to this inherent approximation property, the wavelet transform features a loss-less compression scheme under proper conditions.
FIGURE 4.2 2D wavelet transform following the non-standard decomposition scheme, from resolution level $m_0$ to level $m_0 + 2$.

Additionally, an analysis of the non-zero coefficients can still be performed in order to further downsize the set of coefficients that need to be stored, leading to a *lossy compression*. Thus, some function $f(x)$ that is transformed to wavelet space may be approximated as $f_{\text{comp}}(x)$ with a truncated set of the coefficients.

\[
 f_{\text{comp}}(x) = \sum_{k}^{c_{Mk}} \Phi_{Mk}(x) + \sum_{m = m_0 + 1}^{M} \sum_{k}^{d_{mk}} \Psi_{mk}(x) \tag{4.37}
\]

The most important question is how to select the coefficient to be set to zero. A quite natural choice would be to keep the largest coefficients and ignore the small ones, using some threshold $\tau_{\text{coeff}}$, i.e.

\[
c_{Mk}^+ = \begin{cases} c_{Mk} & \text{if } |c_{Mk}| \geq \tau_{\text{coeff}} \\ 0 & \text{else} \end{cases} \quad d_{mk}^+ = \begin{cases} d_{mk} & \text{if } |d_{mk}| \geq \tau_{\text{coeff}} \\ 0 & \text{else} \end{cases}
\tag{4.38}
\]

In fact, if orthonormal bases are being employed, the very simple threshold filtering shown in Expressions 4.38 produces a globally optimal approximation with respect to the $L_2^2$-norm. Moreover, the error introduced can be expressed precisely as is shown in Equation 4.39 below, where $c_{Mk}^-$ and $d_{mk}^-$ denote the rejected coefficients.

\[
 \| f(x) - f_{\text{comp}}(x) \|^2_{L_2^2} = \sum_{k}^{c_{Mk}^-} |c_{Mk}^-|^2 + \sum_{m = m_0 + 1}^{M} \sum_{k}^{d_{mk}^-} |d_{mk}^-|^2
\]

\[
c_{Mk}^- = \begin{cases} c_{Mk} & \text{if } |c_{Mk}| < \tau_{\text{coeff}} \\ 0 & \text{else} \end{cases} \quad d_{mk}^- = \begin{cases} d_{mk} & \text{if } |d_{mk}| < \tau_{\text{coeff}} \\ 0 & \text{else} \end{cases}
\tag{4.39}
\]

In other words, neglecting the smallest coefficients introduces the least error which is also exactly quantifiable. Furthermore, this property still holds for higher dimensional orthonormal bases. Although Expressions 4.38 and 4.39 do allow the rejection of scaling coefficients $c_{Mk}$ at resolution level $M$, note that this happens rather seldom since these coefficients usually concentrate a considerable amount of the function's energy. Consequently, rejecting such coefficients results in severe approximation errors.
Unfortunately, the simple threshold filtering is not applicable for non-orthonormal bases, i.e. semi- or bi-orthogonal sets of basis functions, although the magnitude-based thresholding may still result in a sufficiently accurate approximation [79]. A proper solution to this problem is proposed in [47]. An incremental computation scheme determines the most significant coefficients along with the accurate approximation error, without requiring an explicit reconstruction, i.e. a wavelet back-transform.

However, all algorithms described in this thesis are open to any orthogonal wavelet bases. Readily available in the system is the Haar basis, of course, as well as many wavelets of the Daubechies family. Hence, threshold filtering according to Expressions 4.38 and 4.39 is applicable and will be heavily used for compression purposes (Section 6.2).

A first example of a compressed 2-dimensional function, i.e. an image, is shown in Figure 4.3. The image has been transformed to level \( M = 7 \) using the Haar basis (middle row) and the Daub10 wavelet (bottom row). Coefficient thresholding has been adapted to preserve 90% (leftmost image on each row), 70%, 50% and finally 30% (rightmost image on each row) of the signal's energy—its computation will be discussed in Section 7.3. However, the coefficients that need to be stored drop to roughly 57%, then 20%, then 8% and finally below 3% in either case. Hence, simple coefficient thresholding may lead to compression factors of 5 and more, while still providing acceptable visual image quality. It may be interesting to note the typical blocking artifacts of the Haar basis observable in the middle row of Figure 4.3.

4.4 HIERARCHICAL REPRESENTATIONS FOR IBMR METHODS

The concept of hierarchy may be explored in many different ways, even in the narrow field of image-based modeling and rendering. For instance, it may be applied to algorithms, to control structures, to user guidance, and of course, to the representation of data itself. This short section, however, will be restricted to hierarchical concepts used for the representation of light field data and alike in the field of IBMR, naturally. A concise and up-to-date summary of hierarchical methods in computer graphics as a whole may be found in [48].

It might be interesting to note that there are other kinds of core data, apart from light field data, that may be represented hierarchically. To give an example, the algorithm presented in [112] operates on a hierarchical representation of a complex environment scene and caches rendered images for any node in this hierarchy. It thus constructs a hierarchical image cache that is used to significantly speed up walkthroughs in the scene.

The notion of hierarchy may be found in several approaches to light field representation and encoding. A small group of them that exploit the wavelet transform and thereupon implicitly generate hierarchical data representations have already been discussed in Section 2.3.4, i.e. the Spherical light fields [61], the Wavelet stream [98] and [71] that discusses interactive rendering of wavelet-projected light fields. Other examples are [83] where the hierarchy of the wavelet transform is used to build a progressive compression scheme and [139] that proposes a representation for Concentric mosaics [115] based on a 3D wavelet transform.
It is understood that a hierarchical data description cannot only be constructed by wavelets. For instance, a hierarchical coding scheme for light field data based on disparity maps is presented in [85] as was already reviewed in Section 2.3.3.

Presumably the most acquainted hierarchy for data representation purposes in the field of IBRM that is not based on wavelets, however, is the hierarchy used to solve for the scattered data interpolation problem as is described in the Lumigraph publication [45]. The hierarchy along with its purpose will be briefly discussed in Section 4.5 below.

However, unlike the system described in this thesis, all the above mentioned approaches compute the hierarchy only after data acquisition rather than as an integral part of it. Moreover, in the case of the LUM system [45] the hierarchy itself, generated during the interpolation process, is not used at all later on.

4.5 SCATTERED DATA INTERPOLATION SCHEMES

The actual tri-variate data items in 4-dimensional data, i.e. the coefficients \( \tilde{x}_{i,j,p,q} \), are given by the inner product of the unknown, continuous and 4-dimensional light field function \( L \) with the duals of the chosen basis functions, as was already pointed out in Section 2.2. This means that the integral in Equation 2.3 on page 21 must be evaluated.
This evaluation, which represents the regularization procedure known as the re-binning step, may be computed up to theoretically any precision by using one of the well-known quadrature formulas [108] in case raytraced imagery is being utilized. This is due to the convenient possibility to evaluate the light field function \( L \) wherever needed when using generated input material.

Regrettably, the preconditions change drastically when real images are being used as input material. Firstly, the light field function cannot simply be evaluated where needed for a preferably precise integration result because the samples are predetermined and fix. Moreover, generally speaking, no valid assumptions about the number of samples as well as their distribution are applicable. As a consequence, the integral in Equation 2.3 is approximated using a Monte-Carlo estimator as is shown below for only one dimension.

\[
\bar{x}_i = \langle L \mid \tilde{B}_i \rangle = \int L(s) \cdot \tilde{B}_i(s) ds
\]

\[
\bar{x}_i = \frac{1}{w_i} \sum_k L(s_k) \cdot \tilde{B}_i(s_k) \quad \text{with} \quad w_i = \sum_k \tilde{B}_i(s_k)
\]  

(4.40)

Such a Monte-Carlo integration is consistent with the splat step of LUM [45] that was briefly reviewed in Section 2.2.3. A clear and brief overview of the light field rendering pipeline including the re-binning or reprojection procedure may also be found in [67].

Note that if the basis functions employed are self-dual, then \( x_{i,j,p,q} = x_{i,j,p,q} \) holds. Moreover, the Monte-Carlo integration features linear time complexity in the number of samples, if basis functions with finite support are being applied. On the downside, each data item gets a fourth component, i.e. the additional weight \( w_i \), that needs to be stored along with the three channels R, G and B.

As was mentioned above, the sample's distribution can in general not be influenced if real imagery is being used as input material. Consequently, they will not be evenly distributed, i.e. regions with no sample values at all will occur. Accordingly, the weight \( w \) of Equation 4.40 will be equal to zero whenever there is no sample in the support region of a specific basis function and hence, the corresponding coefficient \( x_{i,j,p,q} \) will be undefined. It is understood that such data holes must be identified and filled with meaningful data. This task is a special case within the field of scattered data interpolation methods, a well-known approximation problem.

The general scattered data interpolation problem may be stated as follows. Given the data

\[
(x_i, y_i) \in \Omega \times \mathbb{R} \quad i = 1, 2, \ldots, n \quad \Omega \subset \mathbb{R}^k
\]  

(4.41)

find a continuous function \( s \in S(\Omega) \) such that

\[
s(x_i) = y_i \quad i = 1, 2, \ldots, n.
\]  

(4.42)

The integer \( k \) is the number of independent variables. In the light field setting, the data vector's dimension is \( k = 4 \). Furthermore, the integer \( n \) is the number of data points \( x_i \) to be interpolated to, and \( \Omega \) is a suitable domain containing them. \( S(\Omega) \) is the interpolating space, i.e. a linear space of functions defined on \( \Omega \). It might be convenient to think of the data as having been generated by a so-called primitive function \( f_{\text{gen}} \), i.e. 
4.5 SCATTERED DATA INTERPOLATION SCHEMES

$$\mathbf{y}_i = \mathbf{f}_{\text{gen}}(\mathbf{x}_i) \quad i = 1, 2, \ldots, n . \quad (4.43)$$

However, it is important to note that, in general, it cannot be assumed that any information about \( \mathbf{f}_{\text{gen}} \) is available other than the data points themselves.

Many different ways of finding a suitable interpolant \( s \) for many different applications have been proposed in the past. An excellent survey of techniques for the interpolation of scattered data in three or more independent variables with emphasis on breadth rather than depth may be found in [4].

Among all classes of interpolation schemes (tensor product schemes, natural neighbor interpolation, \( k \)-dimensional triangulations, tetrahedral schemes, simplicial schemes, multi-variate splines, etc.) this section will be limited to point schemes only, which refers to interpolation schemes that are not based on a tessellation of the underlying domain \( \Omega \).

Moreover, the special properties of the scattered data interpolation problem in the context of light field rendering further limits the range of applicable solutions. Firstly, the dimensionality is high, secondly, the data points as well as the data holes are gridded, and thirdly, the number of data points can easily grow up to a few millions. As a consequence, all algorithms based on a system of equations are not suitable, because it is highly desirable to only adopt techniques that offer linear time and linear memory space complexity in the number of data points. Additionally, building spatial data structures such as Delaunay triangulations is too expensive.

A solid solution that yields convincing results in the special case of the high-dimensional light field function that is incompletely and irregularly sampled is introduced by Gortler et al., the authors of LUM [45]. This 3-step procedure that has already been touched in outlines in Section 2.2.3 will be reviewed a bit more detailed in Section 4.5.3. Because Gortler’s splat–pull–push interpolation filter is a simplified version of Burt’s hierarchical polynomial fit filters presented in [14] this latter method will be briefly revisited in its 1-dimensional form beforehand in Section 4.5.2. However, the following Section 4.5.1 first starts with the presumably best known method among all scattered data interpolants in a general number of variables. It is the only standard scheme that will be reviewed in this section due to its similarity to Gortler’s interpolation filter.

4.5.1 Shepard’s Methods

In its simplest form, Shepard’s method defines the interpolant \( s \) as is shown in Equation 4.44 below.

$$s(\mathbf{x}) = \sum_{i=1}^{n} w_i(\mathbf{x}) \cdot \mathbf{f}_{\text{gen}}(\mathbf{x}_i) = \sum_{i=1}^{n} w_i(\mathbf{x}) \cdot \mathbf{y}_i \quad \text{with}$$

$$w_i(\mathbf{x}) = \frac{\|\mathbf{x} - \mathbf{x}_i\|^{-p}}{\sum_{j=1}^{n} \|\mathbf{x}_i - \mathbf{x}_j\|^{-p}} \quad (4.44)$$

In this form, the evaluation of \( s \) at a data point \( \mathbf{x}_i \) leads to a division by zero. However, the definition of \( s \) can be extended continuously by the interpolation requirement \( s(\mathbf{x}_i) = \mathbf{y}_i \). The method is a particular example of a convex combination based scheme, since the weights \( w_i \) are non-negative and sum up to one. The interpolant is arbitrarily
often differentiable if \( p > 1 \). Additionally, the first derivatives vanish at the data points if \( p > 1 \). Moreover, the method has the following interesting property:

\[
\min \{ y_i \} \leq s(x) \leq \max \{ y_i \} \quad \forall x \in \mathbb{R}^k \quad i = 1, \ldots, n
\] (4.45)

However, Shepard's method in its unmodified form shows some deficiencies that can be overcome in various ways. To give an example, the primarily global method may be localized by multiplying the basis functions \( w_i(x) \) by so-called mollifying functions \( \mu_i(x) \) that satisfy \( \mu_i(x) = 1 \) and that have local support in some appropriate sense. A possible choice for this functions are the Franke–Little weights [6] as is shown in Equation 4.46 below. The \( R_i \) are suitably chosen radii of circles around the data points that constitute the support of the modified basis functions.

\[
\mu_i(x) = \left( 1 - \frac{\|x - x_i\|^2}{R_i^2} \right)^v \quad \text{where} \quad a^v = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{if } a \leq 0 \end{cases}
\] (4.46)

Another improvement considers the precision of Shepard's method which is only constant in the original form. In order to increase the precision one may interpolate to Taylor expansions instead of function values.

### 4.5.2 Burt's Hierarchical Polynomial Fit Filters

The uniform hierarchical procedure described in [14] is designed for the processing of incomplete image data. It is derived for the 1-dimensional case with subsequent extensions to 2D for the target area of applications, i.e. processing of incompletely or irregularly sampled image data.

The method which is based on a local polynomial approximation in the least squares sense proceeds in the following three steps:

1) **Generation of moment images:**
   
   Firstly, a set of moment images is computed whereby the term image applies to basically any input data field of any dimensionality. Each moment image is an array of moment values computed within a local neighborhood of each input sample point. This neighborhood is defined by a Gaussian–like window function \( W_G(i) \). The moment image of degree \( p \) of some input data \( L(x) \) is defined as is shown below.

\[
L_p(x) = \sum_{i = -\infty}^{\infty} W_G(i) \cdot L(x + i) \cdot i^p
\] (4.47)

Moments are recomputed for a sequence of windows which differ in size by factors of two, leading to a hierarchical pyramid of moment images. It is shown in [14] that a \( p \)-th moment at hierarchy level \( m \) can be expressed as a weighted sum of \( p \) and lower order moments at level \( m - 1 \).

2) **Hierarchical polynomial fit (HPF) filtering:**

As a second step, polynomial surfaces are fit within the local Gaussian neighborhoods of each sample, on each hierarchy level. Best fit polynomials are obtained directly from the local moments. The value of the polynomial at the sample point is taken as the corresponding output sample value.
This fast and uniform computation called HPF filtering generates a whole pyramid of data fields, one for each Gaussian window size. Each level of the pyramid represents a low-pass filtered version of its predecessor and thus, a low-pass version of the source image or data field, respectively. In other words, a multiresolution representation is being generated from primarily irregularly sampled data.

It is important to note that potential holes in the original data are getting smaller with increasing level of hierarchy since the data gets extrapolated by the polynomials within the corresponding Gaussian window.

3) Multiresolution interpolation:
Finally, data in the various low-pass images are being combined to form a smooth surface—at all scales—that passes through the original sample points. Gaps of missing data in the input image are filled such that high frequency information are extrapolated a short distance from the known data while progressively lower frequency components are extrapolated proportionally further. This is achieved by recombining data of each level with its direct predecessor, starting at the maximum level. Hence, large gaps of missing data get interpolated with lowest frequency data whereas small gaps are being filled with higher frequency data.

The scheme as is reviewed in this section is rather complex already in a 2—dimensional setting, even if simple fitting polynomials of degree one are being employed. Moreover, if extended to a 4D, the whole process gets inefficient. The adapted and simplified procedure named splat-pull-push interpolation filter, as is described in the LUM [45] publication, represents the way out in the context of light field rendering.

4.5.3 Gortler's Splat-Pull-Push Interpolation Filter

As is suggested by the name itself, the splat-pull-push interpolation scheme also proceeds in three phases, as is reviewed below. Furthermore, it adopts the core idea of Burt's HPF filtering, i.e. it uses a pyramid–like algorithm to fill in large gaps of missing data with lower frequency data and small gaps with higher frequency data. Moreover, it combines another important concept into the same algorithm, which is a solution for the non-uniformity of the input sampling density that was presented in [92].

The problem caused by non-uniform sampling as is addressed in [92] is solved in the context of image filtering. When obtaining a pixel value through averaging that has been super-sampled with a non-uniform density, one does not want the result to be overly influenced by the regions sampled most densely. The solution presented first computes average values in smaller regions, so-called strata. The final pixel value is then computed by averaging together these strata values, which is not influenced anymore by the number of original sampling values falling in each of the strata. Hence, the non-uniformity of the samples does not bias the processed version. The interpolation filter presented in [45], however, also ensures that the non-uniformity of the input data does not bias the filtered data while generating the lower resolution versions, i.e. while building up the pyramid representation, by following the ideas presented in [92].

During the first phase, called splat, the sampled input data is used to approximate the integral of Equation 2.3 on page 21, by applying the Monte-Carlo estimator shown in Equation 4.40, thereby initializing the pyramid on hierarchy level \( m = 0 \):
Note that the weights will be small or even zero in regions where there is little or no nearby input data, as was already mentioned before.

In the second phase, called pull, coefficients are computed for basis functions at a hierarchical set of lower resolution versions of the input data. More precisely, the lower resolution coefficients are being computed by combining the higher resolution coefficients using some low-pass filter kernel $h$:

$$w_i^{m+1} = \sum_k \tilde{r}_{m+1} \cdot w_k^m$$

$$\tilde{x}_i^{m+1} = \frac{1}{w_i^{m+1}} \cdot \sum_k \tilde{r}_{m+1} \cdot w_k^m \cdot \tilde{x}_k^m$$

The basis functions $\tilde{B}_i^m$ at each hierarchy level are defined by linearly summing together the higher resolution functions, as is shown below.

$$\tilde{B}_i^m = \sum_k \tilde{r}_k \cdot B_k$$

The above formula (Equation 4.49) computes the same result as would the original estimator of Equation 4.48 when applied to wider basis functions. However, it works fine for uniform sampling densities. In order to account for the non-uniformity the weights in the above formula are being limited, thereby applying Mitchell’s reasoning [92], resulting in the formula shown below.

$$w_i^{m+1} = \sum_k \tilde{r}_k \cdot \min(w_k^m, 1)$$

$$\tilde{x}_i^{m+1} = \frac{1}{w_i^{m+1}} \cdot \sum_k \tilde{r}_k \cdot \min(w_k^m, 1) \cdot \tilde{x}_k^m$$

The $\min$ operator is used to place an upper bound on the degree that one coefficient in a highly sampled region can influence the total sum. The value 1 therethrough represents full saturation. The weights thus may be understood as a kind of confidence measure. It is important to note that the $\min$ operator represents a non-linear element in the whole setting.

During the third stage, called push, the lower resolution approximations are used to fill in regions in the higher resolution that have low weight. If a higher resolution coefficient has a high associated confidence, i.e. a weight greater than one, the lower resolution information is fully disregarded. Vice versa, if a higher resolution coefficient does not have sufficient weight, the lower resolution information is blended in. To this end, the lower resolution information must be expressed in the higher resolution basis which is done by upsampling and convolving with a sequence $(h)$ that satisfies the following condition.
\[ B_i^{m+1} = \sum_k h_k \cdot B_k^m \] (4.52)

Using \( h \), temporary values are computed (Equation 4.53) which are then blended with the values at the next level (Equation 4.54).

\[ w_{i, \text{temp}}^m = \sum_k h_{i-2k} \cdot \min(w_k^m + 1, 1) \]
\[ x_{i, \text{temp}}^m = \frac{1}{w_{i, \text{temp}}^m} \cdot \sum_k h_{i-2k} \cdot \min(w_k^m + 1, 1) \cdot x_k^m + 1 \]

(4.53)

\[ w_i^m = w_{i, \text{temp}}^m (1 - w_i^m) + w_i^m \]
\[ x_i^m = x_{i, \text{temp}}^m (1 - w_i^m) + x_i^m \cdot w_i^m \]

(4.54)

It may be interesting to analyze the effectiveness of this 3-stage procedure by first applying it to 2D images. In doing so, the results can easily be evaluated visually. Figure 4.4 below shows a first example of the capabilities of the procedure when applied to a scattered image. The scattered version was first generated by picking random lines and sampling the original image densely along these lines, such that only 50% of the original pixels are known. The hierarchical scattered data interpolation scheme was then applied to this scattered version. The resulting image is shown in Figure 4.4, b). As a matter of course, equivalent results have already been demonstrated in [45].

**FIGURE 4.4** Result of the hierarchical scattered data interpolation scheme from 50% of known data (black areas in image a) represent unknown data.  
a) Scattered version, generated using a line sampling pattern.  
b) Result of the interpolation procedure.  
c) Original image.

The line sampling pattern mimics in many ways the structure of light field samples taken from a hand-held camera, adapted to a 2-dimensional space. In that case, each input image represents a dense sampling of the 4-dimensional light field function along a 2D plane.
The sampling pattern has, however, impact on the quality of the result. The top row of Figure 4.5 below shows the result of the interpolation scheme when applied to a scattered image using the same line sampling pattern again, with only 20% of known data. Clearly, the algorithm gives more homogeneous result when applied to a randomly point sampled image, i.e. white noise, due to the evenly distributed input data, as is shown in Figure 4.5, bottom row.

**FIGURE 4.5** Result of the hierarchical scattered data interpolation scheme from 20% of known data (black areas in images a) represent unknown data).

a.1) Scattered version, generated using a line sampling pattern.
b.1) Result of the interpolation procedure from image a.1).
a.2) Scattered version, generated using a point sampling pattern with clustering.
b.2) Result of the interpolation procedure from image a.2).
a.3) Scattered version, generated using a point sampling pattern (white noise).
b.3) Result of the interpolation procedure from image a.3).
c) Original image.
4.6 CONSTRUCTION OF THE EMRA

The extended multiresolution analysis (eMRA) was designed in order to meet all the requirements that have been assembled and discussed in Section 4.1 on page 51—which are handling of incomplete and adaptive sampling, free choice of an interpolation filter kernel, support for progressive and incremental operations as well as for fast interaction operators, and finally support for data compression.

The eMRA was devised as a unified representation for sampling, interpolation, compression and rendering. In essence, it merges an MRA and a hierarchical scattered data interpolation scheme (HSDI) into one representation. In other words, the hierarchy built during the process of the HSDI is being converted into a consistent MRA while constructing it. To this end, a fully 4-dimensional wavelet transform is being applied, with the additional freedom of an independent interpolation filter kernel.

Figure 4.6 below shows an overview of the representation at construction time including the initial projection path, the subdivision and non-linear fill-in procedure as well as the projection onto difference space. It is important to note that 1D notation for better readability is being used throughout this section. In fact, the representation deals with multi-dimensional multi-channel data, i.e. a 4-dimensional set of data items each of which containing four channels, as was previously mentioned.

The initial data vector \( \mathbf{x}^0 \) is computed by projecting the light field function \( L(s) \) onto a chosen basis. This is accomplished by computing the inner products of \( L \) with the duals \( \tilde{B} \) of the basis functions as is shown in Equation 4.55 below.

\[
x_j^0 = \langle L | \tilde{B}_j^0 \rangle = \int L(s) \cdot \tilde{B}_j^0(s) ds \quad \text{with} \quad (4.55)
\]
\[ L(s) = \sum_{i=1}^{k} L_i \cdot \delta_{s, s_i} \quad (4.56) \]

In other words, \( L(s) \) is given by the set of \( k \) samples at some spatial position \( s_i \) within the support of the basis function \( B \). Inserting the above Equation 4.56 in the previous Equation 4.55 yields the following expression:

\[
x_j^0 = \left\langle \sum_{i=1}^{k} L_i \cdot \delta_{s, s_i}, \tilde{B}_j^0(s) \right\rangle
= \sum_{i=1}^{k} L_i \left\langle \delta_{s, s_i}, \tilde{B}_j^0(s) \right\rangle = \sum_{i=1}^{k} L_i \cdot \tilde{B}_j^0(s_i) \quad (4.57)
\]

In practice, samples of the primal functions \( B \) instead of the duals \( \tilde{B} \) are being used. In case of a box basis, \( B = B \) applies. The duals of the quadra-linear basis—the basis that is actually used in the eMRA setting—are more complex, but the basis functions sufficiently approximate their own duals for our purposes, thus assuming nearly orthogonal decompositions (Section 4.2).

The above Equation 4.57 defines how to first compute the components of the data vector \( x_0 \), i.e. the coefficients \( x_j^0 \), given an initial set of parameterized image data samples. This processing step basically corresponds to the splat procedure (Equation 4.48) of the splat—pull—push scheme (Section 4.5.3).

Along with the coefficients \( x_j^0 \) the weights \( w_j^0 \) are being computed as is shown in Equation 4.58 below:

\[
w_j^0 = \sum_{i=1}^{k} \tilde{B}_j^0(s_i) \quad (4.58)
\]

First these weights are used to perform a normalization step after the initial projection during which the weights themselves are being adapted, too, in order to account for the non–uniformity of the sample distribution.

\[
x_{j, \text{new}}^0 = \frac{1}{w_j} \cdot x_j^0 \quad w_{j, \text{new}}^0 = \begin{cases} w_j^0 & \text{if } w_j^0 \leq 1 \\ 1 & \text{if } w_j^0 > 1 \end{cases} \quad (4.59)
\]

Contrary to the original splat—pull—push scheme, there will be no further need to adjust the weights again if normalized filter kernels are being employed throughout the subsequent hierarchical interpolation procedure.

After the projection of the initial input data set the construction of the hierarchy essentially functions as follows. From the initial data vector \( x_0 \), the hierarchical pyramid is built up through successive application of the projection operator \( P^m \). This operator performs a low–pass filtering with subsampling by a factor of 2 in each dimension on each approximation level \( m \). It recognizes and handles locations with missing data, so-called NO_DATA areas, by utilizing the information stored as the weights \( w_j \). The operator thereby acts as an interpolation operator on a level–by–level basis. The successive filtering
and subsampling stops at a maximum hierarchy level $M$ when all NO_DATA areas have faded.

The ensuing subdivision step starts at level $M$, now applying the wavelet subdivision operator $S^m$, again level by level. As a matter of course, the employment of $S^m$ includes an upsampling operation. On each level completed, an additional, non-linear fill-in operation is performed that eventually restores previously known data in each frequency subband. The corresponding fill-in operator $R^m$ computes the final data $c^m$ at level $m$ from the estimate $\tilde{x}^m$ and the initial projection $x^m$, as is shown in Equation 4.60 below.

$$c^m = R^m(\tilde{x}^m, x^m) \quad \text{with}$$

$$c_j^m = \begin{cases} 
- \frac{m}{x_j} & \text{if } x_j = \text{NO_DATA} \quad \text{and} \quad m < M \\
\frac{m}{x_j} (1 - w_j) + \frac{m}{x_j} w_j & \text{otherwise} 
\end{cases} \quad (4.60)$$

Once the final data $c^{M-1}$ at level $M-1$ is found, it is being projected to wavelet space by using standard wavelet projection operators $P^m$ and $Q^m$. On all lower levels $m < M - 1$, however, detail information is encoded using idem operator $Q^m$, thereby converting the hierarchy into a consistent MRA while constructing it.

Figure 4.7 below adds the complete operator setting employed to the previously shown overview (Figure 4.6).

![Operator setting used for the construction of the eMRA.](image)

It is important to note that it is required to propagate any changes in the eMRA up the hierarchy to all $c^m$, $n > m$ whenever a new $c^m$ is being computed. This is caused by the fill-in operator $R^m$ and its properties. According to Equation 4.60 above it not only adds new information from higher levels of approximation, i.e. lower frequency subbands, but it might also adapt previously known information, i.e. coefficients $x_j^m$, via the blending operation.
If, on the other hand, the blending of coefficients is omitted by adjusting operator $R^m$ as is shown in Equation 4.61 it is still necessary to perform the propagation of changes up the hierarchy. It is the non-linearity of the fill-in operation that makes it impossible to capture the difference information per frequency subband but in the detail coefficients, i.e. to record $T^m d^{m+1} = c^m - \tilde{x}^m$, with $T^m$ being a linear wavelet reconstruction operator. The Expressions 4.62 indicate that it is not possible to arrive at this desired form using the operator $R^m$.

$$e^m = R^m(\tilde{x}^m, x^m) \quad \text{with}$$

$$c^m_j = \begin{cases} \tilde{x}^m_j & \text{if } x_j = \text{NO\_DATA} \text{ and } m < M \\ x^m_j & \text{otherwise} \end{cases}$$

$$e^m = R^m(\tilde{x}^m, x^m)$$

$$= R^m(\tilde{x}^m, S^m e^{m+1} + T^m d^{m+1})$$

$$= R^m(\tilde{x}^m, \tilde{x}^m + T^m d^{m+1}) = T^m d^{m+1} + \tilde{x}^m$$

The propagation of changes up the hierarchy is accomplished by standard wavelet projection operators $P^m$ and $Q^m$.

A second, important yet nice issue to note is the independence of the projection operator $^*P^m$ of the operator setting employed for the projection to difference space. As a consequence, it can be any reasonable interpolation filter, e.g. a Gaussian or a B-spline. As a matter of fact, the results shown in Figure 4.4 on page 73 as well as in Figure 4.5 on page 74 have been generated by applying the operator setting pointed out in this section in 2D and using a Gaussian for the initial projection path. More precisely, a Gaussian filter of tap size 5 was employed for all of the four examples. The maximum depth of the hierarchy was $M = 2$ for the illustrations in Figure 4.4 and Figure 4.5, 3) and $M = 3$ for the examples in Figure 4.5, 1) and 2). In addition, note that interpolation is not only performed in individual image slices but rather across neighboring slices if the scheme is applied in a 4D setting.

The hierarchical eMRA as is presented meets many of the requirements brought up at the beginning of this section. It is capable of handling incompletely and irregularly sampled input data through regularization and the built-in scattered data interpolator, it offers the possibility to freely choose any sound interpolation filter and finally, it conceptually supports data compression due to the wavelet-coded representation—as was mentioned before, compression will be discussed in detail in Section 6.2.

After having constructed the eMRA, the system's core completes the so-called approximation mode (Section 3.3.1). During this first mode, the hierarchical representation is computed as an integral part of the data acquisition process. Throughout the second mode, the oracle mode (Section 3.3.2), the core uses the wavelet-transformed representation to continue the data acquisition process. The mechanisms needed to perform this continual refinement include a—preferably incremental—update operator as well as a
4.7 INCREMENTAL UPDATES

Any new image data may be plaited into the hierarchical representation by exploiting the linearity of the wavelet operator setting. Such an update procedure results in a recurrence relation as is expressed in Equation 4.63 below. New input image data is expressed as $c_{\text{new}}^0$ whereas $\Delta c^m$ denotes the incremental information with respect to $c^m$.

$$c_{\text{new}}^m = p^{m-1} c_{\text{new}}^{m-1} = p^{m-1} (c^{m-1} + \Delta c^{m-1})$$

$$= c^m + p^{m-1} \Delta c^{m-1} = c^m + \Delta c^m$$

Thereupon, it is only the incremental information that needs to be inserted. Note that this increment represents local changes only.

As was mentioned before, the local projection operator needed to perform such an incremental update basically works on 2D slices out of a 4-dimensional data set, instead of the 4D representation as a whole. That is, it should be an operator of complexity $O(N^2)$ instead of $O(N^4)$—which would be the case for a complete transform or back-transform operation, respectively—with $N$ denoting the size of any dimension of some data set. The same argument also holds for the local reconstruction operator, of course. Chapter 5, and Section 5.4 in particular, give a more thorough analysis of these complexity issues and of computational costs, supported by numerous measured results.

4.8 ORACLE MECHANISM

The oracle mechanism itself which decides on the acceptance of each new image sample individually may be designed surprisingly simple. The core module first renders a view that corresponds to the position of the new image data sample. It then computes the difference of input and estimated view. Any difference represents missing information. In case this difference information meets a threshold criterion it is fed into the eMRA using the above mentioned incremental insertion operator—if not, it is discarded.

With $c_{\text{new}}^0$ being a new image data sample and $c^0$ the corresponding reconstructed view, the difference information $\Delta c^0 = c_{\text{new}}^0 - c^0$ is decomposed and inserted using the
procedure shown in the previous Equation 4.63. As a matter of course, the same scheme is applicable to compute \( d_{new}^m \) by using \( Q_{m-1}^n \) instead of \( P_{m-1}^n \).

However, this scheme is applied only if the difference vector \( \Delta e^0 \) meets the threshold criterion, of course. To this end, the system computes the RMSE of \( \Delta e^0 \) and compares it to a specified update threshold \( \tau \), as is discussed in detail in Section 7.5. Additionally, Section 7.3 states the exact formulae used to compute RMSE values and other error measures.

\[
\begin{array}{c}
\text{FIGURE 4.8} \\
\text{a) Reconstructed view of the coarsely sampled light field.} \\
\text{b) Equal view after the construction of the eMRA, completing the approximation mode.} \\
\text{c) New input image.} \\
\text{d) Difference image used for the incremental update during the oracle mode.} \\
\text{e) Reconstructed view after the incremental update operation.}
\end{array}
\]

4.9 VISUAL PERFORMANCE

The construction of the eMRA during the approximation mode as well as the refinement scheme during the oracle mode have been tested on various input data sets. This section, however, discusses the visual performance of the system’s core on two different sets of input images. The equally important computational performance will be commented on in Section 5.4.

The first example illustrated in Figure 4.8 demonstrates all processing steps and both modes of the core module. Image a) shows a rendering right after the initial construction of the light field. The set of pre-rendered images for this light field contains 150 images in total. They are regularly distributed on a grid of 30 (horizontal) times 5 (vertical)
sample locations, resulting in a coarse sampling in the vertical direction. A set of 120 images were randomly selected for the construction of the initial light field of resolution $16 \times 256^2$. The dark regions in image a) show so-called NO_DATA areas, i.e. areas with missing data.

Image b) of Figure 4.8 shows the result after the construction of the eMRA during the approximation mode, as was described in Section 4.6 before. As a matter of course, the viewpoint was not changed as compared to image a). The transformation to difference space was completed to level $M = 1$ using the Haar wavelet basis. Note the weak ghosting artifacts of neighboring views to the left and to the right of the truck.

In general, ghosting occurs during the process of merging together a series of images with non-stationary image regions, either caused by moving objects in the scene and/or by the camera's movements. That is, the truck shown in the example of Figure 4.8 does not necessary cover the same region in all images of the series due to the changing position of the camera. The resulting ghosting artifacts are in essence neighboring views that are blended in as a result of the hierarchical interpolation procedure. A brief discussion of ghosting artifacts in the context of plenoptic modeling may be found in [100] for instance. However, severe ghosting also occurs in other areas of computer graphics dealing with a series of input images, such as image stitching algorithms, for example. A detailed problem description as well as an automatic solution for ghosting elimination in the scope of image stitching is given in [133].

Images c) through e) demonstrate the core's oracle mode. Image d) illustrates the differences between the reconstructed view b) and a new input image c). Using this difference information only to perform an incremental update according to Equation 4.63 and again rendering the same view results in image e), therewith reproducing input image c).

The second example illustrates the visual performance of the incremental update operation alone, as is used during the core's oracle mode. Figure 4.9 a) is a reconstructed view of a densely sampled light field, rendered from the eMRA representation. The light field's resolution was set to $16 \times 256^2$ and it was constructed from initially 250 input images, randomly selected from a stack of 450 pre-rendered images. The transformation to difference space was again completed to level $M = 1$ using the Haar basis.

Image b) is a raytrace of the same view as for image a). Note the differences in high frequencies between these two images. Again, it is only this difference information as is shown in image c) that is used to perform an incremental update operation according to Equation 4.63. Rendering the same view after the update again yields image d) which is not distinguishable from the input image b).

Table 4.2 gives a compilation of the parameters that precisely describe the two light fields shown in the above examples. Likewise, Tables B.1 and B.2 in Appendix B summarize all relevant parameters of the two sets of input images these two light fields are generated from.

The eMRA representations of the two test examples both employed the Haar basis for the transformation to wavelet space. Note that all algorithms during both modes, i.e. all procedures that construct or alter the data representation in any way, do work with any orthogonal wavelet basis. The system as is includes several bases that are readily available to be used, which are a collection of the Daubechies family of wavelets in addition to the Haar basis. For the examples shown, the Haar basis was chosen mainly because of its lower computational costs during transformation operations due to its highly compact support,
FIGURE 4.9  Incremental update operation using the Falls light field.
   a) Reconstructed view before the update.
   b) New input image.
   c) Difference image used for the incremental update.
   d) Reconstructed view after the update.

as compared to wider basis functions. Additionally, since the interpolation operator is independent of the wavelet coding, the Haar basis with its rather poor interpolation capabilities does not influence the visual quality at all—which is, of course, only an admissible statement as long as no compression mechanisms are active at all (Section 6.2).
Moreover, note that the input imagery for both examples, i.e. the Truck light field as well as the Falls light fields, has been generated using the raytracing module of the ALIAS|Wavefront Studio|Tools [ALIAS©Tools]. The models themselves are demo scenes bundled with this software package [ALIAS©Models].

Further examples underscoring the visual performance based on real input imagery may be found in Chapter 7, e.g. Figures 7.8 through 7.11 demonstrating crudely and more densely sampled light fields and the results of the core's approximation mode, Figure 7.12 showing the effect of the HSDI scheme in particular and Figure 7.13 again illustrating the impact of a single local update operation.
Seite Leer / Blank leaf
Throughout its first mode, the system’s core aims at finding a sound estimate of the light field data assembled from an initial set of input images. To this end, it constructs the eMRA according to the procedures set out in the previous Chapter 4, resulting in the wavelet–transformed multidimensional representation.

By switching to the core’s second mode (Section 3.3.3 on page 49), the approximation is getting continuously refined, using the incremental update scheme. Every update cycle thereby involves 1) a reconstruction of a given view, based on the data known so far, 2) a comparison of this reconstructed view with a new input image and finally 3) an eventual incremental update operation. As a matter of course, both the reconstruction as well as the update operation need to directly work on the wavelet–transformed data—a complete back–transform in order to perform the readout and potential updates with an ensuing complete re–transform is computationally much too costly, as is briefly highlighted at the beginning of the following Section 5.1.

The local operators as they are introduced in this chapter demonstrate a possibility to perform both operations, i.e. reconstructions and data insertions, directly in the wavelet domain by working on the coefficients themselves rather than on back–transformed data items. Section 5.1 first introduces the concept these local operators are based on. The succeeding sections then point out the local reconstruction operation used for rendering (Section 5.2) and the local projection operation used for data insertion (Section 5.3). Finally, the computational performance is commented on in Section 5.4.

5.1 CONCEPT

In general, the computational complexity of a wavelet transform for an \( n \)-dimensional data set of size \( N \) in each dimension grows with \( O(N^n) \)—which is \( O(N^d) \) in the present case. Clearly, a complete back–transform with a subsequent re–transform for an eventual update operation needs to be avoided.
Both the local reconstruction operator used for rendering and the local projection operator needed for incremental updates do minimize the computational complexity of a complete update cycle on the representation chosen. In essence, they are of $O(N^2)$ because they basically work on 2-dimensional slices through the 4-dimensional and transformed data set, as was previously mentioned. Such 2D slices correspond to single images, as is deepened in Section 5.1.5. Before, the essential idea of the local operators (Section 5.1.1), two possible strategies for their realization (Section 5.1.2) and the actual implementation (Section 5.1.3) plus an example in a 2-dimensional setting (Section 5.1.4) are being discussed. The section closes with a brief discussion of related concepts that might be found in the literature (Section 5.1.6).

5.1.1 Fundamental Idea

The fundamental concept behind the idea of such local operators that directly work on coefficients is the following. For either operator, identifying the subset of all coefficients or basis functions that are actually affected by any operation leads to a comparably small number of coefficients, or a small number of basis functions, respectively. The favored simplification in terms of computational complexity may be achieved by only involving such a subset while processing and leaving all other coefficients and basis functions untouched. Identifying such a subset is allowed by the wavelets’ properties of being well localized both in frequency and time—in contrast to the Fourier transform, for example (Section 4.3.3 on page 59).

The size of a subset of coefficients affected by an operation, i.e. the size of the support of a reconstruction or edit operation, is influenced by 1) the wavelet basis, 2) the dimensionality of the data set and 3) the maximal depth $M$ of the hierarchy. The efficiency of a local operator, in turn, is guaranteed by the local support of the operation and secondly, by the performance of the wavelet transform algorithm itself, i.e. the fast wavelet transform (FWT) in particular.

5.1.2 Viable Strategies

From a mathematical point of view, there are at least two different ways how to construct local operators, i.e. how to determine the local support for either operation. One follows the perception of function approximation, the other rather uses an algorithmic point of view based on the FWT as is set out below.

1) Based on function approximation:

Any function $f(x)$ may be written as a linear combination of a coarse approximation of its own and additional detail information on a series of different levels of approximations, resulting in a hierarchical representation (Section 4.3.2). The coefficients $c_{mk}$ and $d_{mk}$ in the expression below are determined by the inner products of the function $f(x)$ with the corresponding wavelet basis functions.

$$ f(x) = \sum_k c_{mk} \phi_{mk}(x) + \sum_{m = m_0 + 1}^M \sum_k d_{mk} \psi_{mk}(x) $$  \hfill (5.1) 

Given such a representation, a local operator might reconstruct the function $f(x)$ at a single position—or edit it at a sole position. To this end, the above sum in Equation 5.1 needs to be evaluated, with just considering the active basis functions at
the desired position, completely neglecting all other basis functions. Naturally, these active scaling and wavelet functions must be evaluated before adding up the sum.

2) Based on the FWT algorithm:
Any implementation of the fast wavelet transform algorithm needs nothing but the coefficients of the two sequences \((h_k)_{k \in \mathbb{Z}}\) and \((g_k)_{k \in \mathbb{Z}}\) of the 2-scale relation for scaling and wavelet bases (Equation 4.18 and Equation 4.24 in Section 4.3.2). These sequences combine basis functions of different scales in order to obtain new scaling and wavelet basis functions. The FWT, however, directly uses the coefficients of these sequences to compute the convolutions with the data. Note that the coefficients of \((h_k)\) and \((g_k)\) are not necessarily samples of the scaling or the wavelet functions themselves. A more thorough introduction into the FWT and its implementation may be found in [79, 118] for instance. A local reconstruction or edit operation in a transformed data set might be performed by first identifying the subset of coefficients lying in the support area of the filter sequence. This, of course, needs to be done on all levels of the hierarchy and in every single frequency subband of each level. The actual local operation might then only involve the thus determined subset of coefficients.

The local operators developed in the course of this thesis try to tag the support area of the filter sequence for a desired operation in the transformed data set, therethrough following the second of the above mentioned strategies.

5.1.3 Implementation
A region of support is definable due to the basis functions’ construction using binary dilations and dyadic translations of some single functions \(\phi\) and \(\psi\) (Section 4.3.2). Hence, given a specific wavelet basis and an arbitrary position in the data field to be filtered, the area of support may be established as follows. The so-called active area \(A(m, h, s_h, j_h)\) on level \(m\) of the decomposition pyramid for a filter sequence \((h)\) of size \(s_h\) centered at position \(j_h\) is bounded by

\[
A(m, h, s_h, j_h) = \left[ j_{h,m} - \left(\frac{s_h}{2}\right) \cdot 2^m, j_{h,m} + \left(\frac{s_h}{2}\right) \cdot 2^m \right],
\]

(5.2)

where \(j_{h,m}\) is the position of \(j_h\) on approximation level \(m\), which is given by

\[
j_{h,m} = \frac{j_h}{(2^m)}.
\]

(5.3)

Note that the above defined bounds are valid for a point operation at location \(j_h\) in a 1-dimensional setting. In fact, the active area is of the same dimensionality as the data field itself, i.e. a 4D volume in the present case. Moreover, in case a bigger region of data is to be affected by a local operation—instead of a single point, e.g. a rectangle area—the corresponding active area may be composed through combination of areas given in the above Equation 5.2.

It is important to further note that the bounds described in Equation 5.2 may slightly vary depending on a specific filter’s tap-size \(s_h\), i.e. whether it is even or uneven, and how its support is centered. The actual algorithm in use employs offset values defined per filter sequence \((h)\) individually to apply corrections if needed.
In every situation where a finite data field of some dimensionality is being convolved with a filter of a given size, the question of how to deal with the field's boundaries will arise. Likewise, the same problem also occurs when applying wavelet transform algorithms. Among a few possible solutions—end–point interpolating wavelet basis, data mirroring, i.e. symmetric extension at the boundaries [33]—the implementation in the system follows the wrap–around strategy where each boundary in each dimension virtually connects to its opposite. Consequently, an active area will continue at the opposite end of a data field, if the position \( j_{h,m} \) lies close enough to its bounds. Note that such a torus–like addressing of data values needs to be done on each level \( m \) of the decomposition pyramid. Examples of such situations are illustrated in Figure 5.1, which is commented on in detail in Section 5.1.4.

As far as an actual implementation is concerned, a wrap–around strategy may be easily realized using a modulo operator. Equation 5.4 below shows how to compute a position \( j_{m,\text{wrapped}} \) that is guaranteed to lie within the valid bounds of a 1D data set on level \( m \), with \( N \) being the size of the original untransformed field of data. Simply speaking, any position \( j_m \) that would fall beyond the valid bounds of a data array gets automatically mapped back to its beginning.

\[
j_{m,\text{wrapped}} = j_m \mod (N/2^m)
\]

Identifying active areas according to Equation 5.2 which are further depending on the dimensionality of the setting and on the type of edit operation, e.g. update or extraction of a point, a line or editing of a slice through a higher–dimensional volume etc., is the first processing step to be performed. The thus selected subset of coefficients are then accessed for the actual processing task. It is important to note that the corresponding local operator leaves all others perfectly untouched.

All dimensions of an active area may be processed sequentially, due to separability of the wavelet transform. Any multi–dimensional data field can be filtered following the tensor product strategy, i.e. all dimensions can be transformed independently, one after another. As a matter of course, this property also holds for the active area. Whether the standard or the non–standard decomposition scheme (Section 4.3.5) is followed does not affect the principle itself. As far as an implementation is concerned, it is easier to follow the non–standard scheme, since the support areas on all approximation levels in all frequency subbands are of the same regular shape, which is not the case for the standard decomposition scheme (Figure 4.1 and Figure 4.2).

The implementations of both local operators, the local reconstruction as well as the local projection operator, are all developed from a single core routine. This procedure applies an arbitrary wavelet filter to a 1–dimensional data field—or its transpose, depending on the corresponding parameter setting. It follows the wrap–around strategy by properly incrementing any pointers referring into the 1D data field it is working on. This algorithm listed in Appendix A is an enhanced version of the partial wavelet transform (PWT) algorithm that may be found in [101]. The necessary adaptation for a local operator basically parameterizes the bounds of the outer loop of the transform code section, i.e. variables \( i \) and \( j \) in Appendix A, according to the boundary calculations of the active area. Moreover, the variables \( i\text{off} \) and \( j\text{off} \) are the previously mentioned offset values needed for the proper centering of filters and eventual adjustment of bounds.
The following sequence of program code similar to the C++ language may give an impression of a routine that performs a local point reconstruction in a 1-dimensional data set. The variable local_pos corresponds to some \( j_{h,m} \) and clearly, pos stands for \( j_h \). The variables \( a \) and \( b \) denote the bounds of the active area, according to Equation 5-2. Note that the shift operators >> and << are nothing but an efficient and convenient way of computing one or more multiplications by a factor of 2, or integer divisions by 2, respectively. Also note that the code sequence below does not include the wrap-around mechanism in order to reduce the complexity of the example.

```c
while (level>0)
{
    local_pos = pos >> level;
    level--;

    size = orig_size >> level;
    half_size = size >> 1;

    a = local_pos - (ncof/2)<<level;
    b = local_pos + (ncof/2)<<level;

    for (j=a, i=2*a-1; i<=2*b-1; i+=2, j++)
        for (k=1; k<=ncof; k++)
            workspace[i+k] += filter_seq_h[k] * data[j];
        workspace[i+k] += filter_seq_g[k] * data[j+half_size];
    end for
    end for

    for (j=2*a-1; j<=2*b; j++)
        data_new[j] = workspace[j];
    end for
}
end while
```

Note that the PWT routine as is listed in Appendix A may be readily used for the construction of higher-dimensional wavelet transforms, based on the FWT algorithm. When invoked in the correct order with the proper references to the right portion of a data field with two or even more dimensions, it computes a wavelet transform or back-transform following the non-standard decomposition scheme.

### 5.1.4 Example in a 2D Setting

Figure 5.1 finally shows examples of active regions for a local point update operation in a 2-dimensional data field. Note that although the wavelet transform is applied to each data channel of image a) separately they are merged together again for visualization.

Images b) and c) of Figure 5.1 show wavelet-transformed versions of the original image a), using Daubechies-4 (b) and Daubechies-8 wavelets (c). Both transformations have been completed up to hierarchy level \( M = 3 \).

Image b) of Figure 5.1 visualizes the active regions for a local update operation on all three levels for the red dot inside the red circle in the upper left corner of image a), additionally pointed at by the dark arrow. The active areas in all frequency subbands on the first two levels are clearly identifiable. The third level shows a wrap-around situation in both directions of the 2D data set.
FIGURE 5.1 Wavelet filter support areas for a local point update operation in a 2D setting.

a) Original image.
b) Transform to level 3, using a Daubechies-4 filter.
c) Transform to level 3, using a Daubechies-8 filter.

Image c) of Figure 5.1, on the other hand, shows the active regions for a local update operation for the yellow dot inside the yellow circle in the lower right corner of image a), also marked by the light arrow. The sizes of the active regions are clearly larger as compared to the ones in image b), which is due to the double tap-size of the Daubechies-8 versus the Daubechies-4 wavelet. Moreover, a wrap-around situation in again both directions of the 2-dimensional data field already occurs on the second level. On the third level, the local update operator automatically switches from local to global filtering on that very level since the active areas already fill out the whole subband.

Note that the thin white lines in images b) and c) have been added for better visualizing the individual subbands. They are not part of the transform nor are they an artifact of any other algorithm. The same holds for Figures 5.3 and 5.4 later on in this chapter.
5.1 CONCEPT

5.1.5 Use of Parameterization

The system's core utilizes either of two alternative settings in order to parameterize new input imagery—which are the cylinder-plane parameterization and the sphere-plane parameterization, as was introduced in Chapter 3.

Parameterization of new input images is done by shooting rays originating at the image's center of projection (COP) and passing through an individual pixel on the image plane. Intersecting these rays with the parameterization objects, i.e. the cylinder or the sphere and the plane, yields the desired four ray parameters (Section 2.2.2 and Section 3.2). As a matter of course, reconstructing a view, i.e. rendering, may be accomplished in the very same way by reading from the data instead of updating it. Such update and reconstruction operations may therefore be subsumed as ray-based operations.

If the data is transformed to difference space as is the case after completion of the core's approximation mode, a ray-based operation makes it necessary to find the active area for each single ray, i.e. for every data item a ray is pointing to. Since these locations are not necessarily all direct neighbors in the 4D data set, the resulting collection of active areas is quite complex in terms of cardinality and shape of the subvolumes.

If, however, the COP of a camera is placed at a sample location \((\phi, h)\) of the cylinder, or at \((\phi, \theta)\) of the sphere, respectively, any image represents a 2-dimensional slice through the 4-dimensional data set, since two out of four parameters are fixed. Hence, rendering a new view means reconstructing a complete 2D plane. Likewise, inserting new image information means editing a 2D plane. In contrast to the ray-based operations, such update and reconstruction operators that work on slices through the 4D data set may be classified as slicing operations.

As a consequence, identifying the active area for a local operation based on slicing is equivalent to determining the support area for a 2D plane in the higher-dimensional data set. Moreover, the fact that slicing operators work on planes is the fundamental reason why such local operators are of \(O(N^2)\) instead of \(O(N^4)\). A more detailed discussion of their computational complexity will be given in Section 5.4.

All local operations the system's core performs on the transformed representation in the course of the oracle mode are in fact operations based on slicing. For ease of notation, the first two parameters \((\phi, h)\) of the cylinder-plane parameterization or \((\phi, \theta)\) of the sphere-plane parameterization, respectively, will be referred to as \(x\) and \(y\) throughout the rest of this thesis. Likewise, the remaining two parameters caused by the intersection with the \(st\)-plane will be referred to as \(z\) and \(t\). Hence, rendering an image means reconstructing a complete \(zt\)-plane for a fixed pair of \((x, y)\) coordinates.

It is important to note that the system automatically switches to the set of local operators based on slicing as soon as the transformation to wavelet space is completed, i.e. as soon as the core quits the approximation mode. If, on the other hand, local reconstruction operations, i.e. rendering, or even local projection operations are being performed before the construction of the eMRA, and therefore before the transformation to difference space, one might freely choose to apply ray-based operators or the set of operators based on slicing. Referring back to Figure 4.8 on page 80 again, image a) was generated using a ray-based reconstruction operator, whereas images b) and c) were computed from the transformed data set by a local reconstruction operator based on slicing—just as images a) and d) of Figure 4.9 on page 82.
Additional and complementary information on ray–based operations versus slicing, such as implications of the latter method, filtering according to the primal quadra–linear basis (Section 4.6), etc., will be given in Section 6.1.3.

5.1.6 Comparable Work

Ideas similar to the local operators as developed in the course of this thesis may be found rather scarce, especially in the field of computer graphics. Nonetheless, the few relevant publications are briefly reviewed below.

The progressive inverse wavelet synthesis (PIWS) algorithm introduced in [139] represents an idea close to the concept of local operators. The scheme allows for random access and partial decoding of wavelet compressed 3–dimensional data. The partial reconstruction is based on inverse lifting operations [34, 124]. The computational overhead for such a reconstruction operation is reduced to a minimum by only accessing and decoding the affected portion of the data, too. However, the PIWS algorithm is designed for reconstruction, i.e. rendering, only. A comparable operator for updating the data is not available. Besides, with the algorithm being designed for generating views out of the 3D data representation called concentric mosaics [115], the dimensionality of the problem is lower than in the case of light field data.

The possibility to only involve small parts of wavelet projected data for an inverse transform, i.e. data reconstruction, may also be found in the field of volume rendering. In [51], for instance, wavelet–transformed volume data is partially decompressed and rendered on demand. The filtering is again based on lifting steps, using the wavelet transform algorithm for integers [15]. Also, it is only the back–transform that is being sped up by accessing a subset of the data—which comes at no surprise since the main interest in volume rendering applications is the fast generation of arbitrary views of possibly very large data sets. However, the important and grave difference to the concept of local operators working on wavelet–coded data as was introduced in Section 5.1.3 is the block–based strategy employed in [51]. The transformation algorithms are not applied to the data set as a whole but to individual small blocks of the subdivided data. Hence, active areas do not need to be identified at all. Once the relevant portion of the volume data, i.e. the set of blocks needed for a reconstruction, is determined, a complete back–transform is applied to each block individually. As a matter of course, this strategy is completely independent of the wavelet transform and its localization properties. That is, the same scheme is also functional with any other transformation—or with no transformation at all.

The same block–based approach is also followed in [29] again applying the lifting method to each block individually. Contrasting the above discussed design especially developed for the use in volume rendering applications, the scheme presented in [29] is a hierarchical representation for gridded floating point data of basically any dimension, due to a clever block reordering after each level of transformation. The selection of data subsets to computationally efficiently handle partial data reconstruction requests is done by preselecting the relevant data blocks, too. Again, the potential of this idea is not further pursued to also perform fast update operations. However, the scheme presented in [29] has been extended and integrated in other toolkits, such as a commercial data analysis package or a visualization toolkit in order to enable progressive data access to large, regular gridded data sets of any kind [30].
5.2 LOCAL RECONSTRUCTION (RENDERING)

A local reconstruction operation is the process of back-transforming a subset of coefficients, i.e., the coefficients located inside the active areas, in order to have the untransformed data readily available. In the 4-dimensional setting, locally reconstructing a 2D slice means restoring an image which is the necessary operation for rendering.

The type of reconstruction operation, e.g., recovering a point, a line, etc., has great impact on the size and shape of the active areas—and consequently on their level-wise processing as well. Fortunately, the given scenario in 4D does not represent some generic case. In fact, the 2D slice out of the 4D data set is an axis-aligned zt-plane for some fixed pair of (x, y) coordinates. Accordingly, the following discussions as well as the examples given do not cover the general but analogous cases.

The processing of a local reconstruction of any type, however, always follows a similar scheme. In doing so, the operator cycles through the following steps, starting from the maximum transformation level \( M \).

1. set current level to \( m = M \);
2. identify active areas on the current level \( m \);
3. successively apply reconstruction filter in all dimensions;
4. set new current level to \( m = m-1 \);
5. if \( (m <> 0) \) go back to step (2);

Note that in order to not negate the complete eMRA stored in some data structure, it is imperative to carry out the processing for a local reconstruction operation in a separate buffer. This additional buffer stores all intermediate results on all levels of approximation, finally converging to the desired result of the reconstruction on level \( m = 0 \).

The next Section 5.2.1 first points up the above processing sequence in two dimensions, before moving on to the 4-dimensional case.

5.2.1 2-dimensional Case

Figure 5.2 below illustrates the general processing scheme for a local reconstruction operation, applied to a point reconstruction in a 2-dimensional data field. Evidently, the above mentioned step (3) needs to be completed for two dimensions, i.e. the x- and the y-direction.

![Diagram of local reconstruction process](image)

**FIGURE 5.2** Individual processing steps for a local point reconstruction from a 2D data set, transformed to level \( M = 2 \).
Note that any implementation must take good care of buffer handling, as is indicated in Figure 5.2. The first filter step on the maximum level $M$ reads data exclusively from the original data structure (reddish areas) that stores the eMRA and writes the first intermediate results to an additional work buffer (greenish areas). The second filter step on the same level reads from and also writes to this buffer. On any other level $m > M$, the first filter step reads data from the work buffer of just one frequency subband. All other subbands are read from the original data structure. However, what does not pose a rough problem in 2D gets a lot more complicated in higher-dimensional settings since the number of filter steps per level, and with it the number of intermediate results, changes drastically.

Figures 5.3 and 5.4 each show an example of a local point reconstruction operation from a wavelet transformed representation of an image. The rightmost image of both illustrations, respectively, displays the transform to level $M = 2$ using the Haar basis in Figure 5.3 and to level $M = 3$ employing the Daubechies–10 basis in Figure 5.4. Again, note that the data set’s channels are merged together for visualization, although the wavelet transform is applied to each channel separately.

The red dot in image a) in each case at which the white arrow additionally points to marks the location to be locally reconstructed. The images b) display the content of the additional work buffer, storing the intermediate results as well as the final point reconstruction. A simple difference analysis of the original pixel and its local reconstruction proofs the numerical correctness of the results.

Moreover, it might be worth noting the wrap–around situation occurring in all dimensions and on all levels in image b) of Figure 5.4—a state that rises quite frequently and therefore needs to be handled automatically.

5.2.2 4-dimensional Case

Rendering an image from the eMRA representation requires to locally reconstruct a complete axis-aligned $zt$-plane in the 4D data set as was pointed out before. Reconstructing a complete $zt$-plane means that global filtering in these two dimensions, i.e. both the $z$- and the $t$-direction, must be applied. As a consequence, active areas in $z$ and $t$ cannot be identified. In the $x$ and $y$ dimension, in contrast, active areas can be determined. The
COP of the virtual camera corresponding to a desired view defines some sample location \((x_0, y_0)\) for which the support areas in 4D may be resolved in a similar way to a point reconstruction in a 2-dimensional setting. Thus, the processing steps of the general scheme for a local reconstruction operation may be readapted for the 4-dimensional case as follows.

1. set current level to \(m = M\);
2. identify active areas in \(x\) and \(y\) according to the position of \((x_0, y_0)\) on the current level \(m\);
3. successively apply reconstruction filter in \(x\)- and \(y\)-direction locally and in \(z\)- and \(t\)-direction globally;
4. set new current level to \(m = m-1\);
5. if \((m <> 0)\) go back to step (2)

The following Figure 5.5 illustrates the above procedure for the last filtering steps from approximation level \(m = 1\) back to \(m = 0\) for the sample location \((x_0, y_0)\). It is important to understand that the \(zt\)-axis represents both the \(z\) and \(t\) dimension. The figure thus mimics kind of a 4-dimensional drawing. A line aligned to this \(zt\)-axis then corresponds to a complete \(zt\)-plane in true 4D.

What Figure 5.5 nicely shows is the collapsing of active areas in any dimension after filtering and up-sampling in the respective dimension. The support area collapses to the single coordinate at which the active area was centered, i.e. \(x_0\) and \(y_0\) in Figure 5.5.

![Figure 5.4](image.png)

**FIGURE 5.4** Example (2) of a local point reconstruction in 2D.
a) Original image (red point marks spot to be locally reconstructed).
b) Work buffer with intermediate results (white arrow marks desired result).
Rightmost image: Transform of image a) to level \(M = 3\), using the Daubechies-10 basis.

![Figure 5.5](image.png)

**FIGURE 5.5** Filtering steps for a local reconstruction operation of a single \(zt\)-plane in 4D, from approximation level 1 to 0, at location \((x_0, y_0)\).
Due to the collapsing of support areas while applying local filtering in the respective direction, it is, in terms of computation time, crucial to first apply the reconstruction filter in a dimension with identified active areas. As is also clearly visualized in Figure 5.5, it is only one single $zt$-plane that is left for further processing after the $x$- and $y$-direction have been completed.

It is, however, important to note that collapsing does not occur but during the last processing steps from level $m = 1$ back to level $m = 0$. A fact, that may also be discovered in images b) of Figures 5.3 and 5.4 where the active areas for the last processing step collapse to just a line of pixels. This last filter operation produces a sole short line of pixels including the final result point—a small line that is hardly noticeable and therefore pointed to by the white arrow.

Nevertheless, taking advantage of the effect of collapsing support areas is important regarding minimizing the overall computation time because the transformation from level $m = 0$ to $m = 1$ or vice versa dominate the overall transformation costs, as is discussed in greater detail in Section 5.4.

### 5.2.3 Caching Mechanism

Despite of a local operator employed for rendering, reconstructions from the 4-dimensional eMRA representation remains a computationally intensive task—the computational performance of the local operators and the benefits they achieve are commented on in Section 5.4. Nonetheless, a suitable caching mechanism may still be a helpful extension to allow for just-in-time (JIT) rendering.

Probably every rendering pipeline bears more or less potential for caching strategies on many different levels or in different states of the pipeline. In the context of volume rendering out of wavelet-transformed 3-dimensional data, for instance, caching areas may be set up for (1) decompressed blocks of volume data, for (2) completely assembled 3D textures and/or for (3) readily available texture data in the memory of a graphics subsystem. It is such a 3-leveled cache mechanism that is employed in the system described in [51].

Likewise, three different caching levels may be utilized in order to improve on the rendering performance in the context of light field rendering from a wavelet-transformed representation, too, as is described in [98].

A simple yet very effective method, however, is the proper caching of already generated views, i.e. reconstructed images. The highest-level caches in [51] as well as in [98] are in fact of this type. As a matter of course, the same idea may be applied to the present context and is in fact integrated in the system described in this thesis.

The implemented caching system operates a small list of directly accessible views that have been newly reconstructed. This list is being queried before any new view is rendered using the local reconstruction operator. If the cache misses, the system reconstructs the view which is then also stored in the cache area, of course—in case of a hit, the view is instantly available.

The size of the cache is definable by setting the cache radius $r_c$ — a parameter that may be specified by a user at any time. It defines half of the side length of a squared area, with respect to the sampling distance in the $xy$-parameter space, which is the space the cache area exists in. This squared area is always centered over the $(x, y)$ coordinates that correspond to the COP of the current view. The cache system's configuration then allows to
buffer all views corresponding to sample locations \((x_i, y_j)\) lying inside the area. The maximum number of entries that may be stored in the cache, i.e., its size, is given by \((2 \cdot r_c)^2\).

Figure 5.6 a) illustrates a squared cache area in the \(xy\)–parameter space, with the cache radius set to \(r_c = 2\). It is centered over some current COP marked by the black cross. The set of green dots represents all potential entries of the cache structure. That is, if an image corresponding to one of these green dots was reconstructed heretofore it is being buffered in the cache. Note that views corresponding to some coordinates \((x_i, y_j)\) not coinciding with a sample location on the grid—as is the case for the black cross in Figure 5.6 a), for instance—are generated through appropriately blending together the views of the neighboring sample points, according to the quadra-linear basis employed (Section 6.1.3).

If the current viewpoint is moved as is indicated in Figure 5.6 b), for instance, the cache area is displaced accordingly. As a consequence, some entries become invalid (reddish dots) while others (black dots) join in. To be more precise, if a view corresponding to one of the sample locations highlighted with a black dot is being reconstructed and needs to be buffered in the already full cache, it is the oldest of the entries highlighted with a reddish dot that will be replaced.

A reasonable value for the cache radius parameter might be \(r_c = 2\) as is the case in Figure 5.6. The interval of valid values, however, is given in Expression 5.5 below. The choice of a large value for \(r_c\) will allow for true JIT rendering once the cache has been populated, at the expense of more memory needs, of course. However, compared to the memory needs of the data structure storing the eMRA additional cache memory may be neglected.

\[
0 \leq r_c \leq \min \left( \frac{x_{\text{max}}}{2}, \frac{y_{\text{max}}}{2} \right)
\]  

(5.5)

Note that any incremental update of the eMRA might invalidate entries buffered in the cache structure, which then have to be removed from it, of course. Moreover, if the current viewpoint is moved close to one of the boundaries of the \(xy\)–parameter space it must be guaranteed that the cache area always remains inside the valid intervals of \(x\) and/or \(y\).

![Figure 5.6](image)

**FIGURE 5.6** Squared cache area of radius \(r_c = 2\) in the \(xy\)–parameter space.

- a) Cache area and its potential entries, centered at location marked by the black cross.
- b) Cache area, moved from its previous (reddish cross) to a new location (black cross).
5.3 LOCAL PROJECTION (INCREMENTAL UPDATE)

Being the counterpart of the local reconstruction scheme needed for rendering, the local projection operation is the process of incrementally inserting new image data into the wavelet-transformed, 4-dimensional eMRA representation.

Since a local projection is, in essence, the inverse operation of a local reconstruction, most of the explanations given in the previous Section 5.2 do also hold for the local projection operator. Most importantly, the 2D slice in the 4D data set affected by an incremental insertion of new image information is also not arbitrarily oriented, but again an axis-aligned $zt$-plane for a fixed pair of $(x, y)$ coordinates.

Just as is the case for the local reconstruction, the processing of a local projection of any type also follows a generic scheme. But, in contrast to the former case, the projection operator starts from approximation level $m = 0$, performing the following steps.

1. set current level to $m = 0$;
2. identify active areas on the current level $m$;
3. successively apply projection filter in all dimensions;
4. set new current level to $m = m + 1$;
5. if ($m <> M$) go back to step (2);

The additional buffer needed for the rendering operation is also required for the incremental update operation. As a matter of course, it is the same buffer that can be reused for both operations since the two schemes are not being processed in parallel. It is basically the elemental rule specified in Section 4.7, i.e. the recurrence relation expressed in Equation 4.63 on page 79, that asks for an additional buffer in order to perform an incremental update. According to that relation, any new data to be inserted must first be transformed to the same level of approximation as the master data already is, before it can be locally added. In other words, the above processing scheme is being executed using the additional buffer with initially nothing stored but the new data to be inserted. The transformed data, i.e. its portions inside the active area in every frequency subband, is then added to the master data which can be done on the fly on a level-by-level basis.

An implementation of the above scheme for a local projection may be realized with just one additional buffer—an insight, that is not obvious at all, because intermediate results must be partially cleared in order to compute numerically correct results. The effect making this corrective necessary as well as the buffer clearing strategy integrated in the local projection process resolving it will be discussed in greater detail in Section 5.3.3.

The next Section 5.3.1, however, illuminates aspects of the local projection sequence, again in a 2-dimensional setting before proceeding to the 4D case in Section 5.3.2.

5.3.1 2–dimensional Case

The illustrations given in Section 5.2.1 all are, in a sense, also valid for the case of a local projection operation. That is, the active areas on the different approximation levels look analogous. An example of a local point update in a 2D setting was given in Figure 5.1 on page 90 which may be consulted for comparison.

The chief difference, however, is the inverse procedure. While a local reconstruction works in a top–down manner from approximation level $m = M$ down to level $m = 0$, the local projection runs bottom–up, from level $m = 0$ up to the maximum level of transformation. There are basically two major consequences to be taken notice of. Firstly, the handling of the additional buffer that is necessary in both cases is quite different.
During reconstruction, reading of data occurs intermixed from the data structure storing the original eMRA as well as from the additional buffer; writing is done to the buffer exclusively (Section 5.2.1). Contrary to that, the local projection of new data is performed using the additional buffer solely until the transformation to level $m = M$. The actual insertion of the new data into the eMRA representation may be done after the full transformation. A smarter solution performs the transfer of all processed frequency subbands upon completion of each level immediately.

Secondly, an even more important consequence is the fact that the projection filter does not scale down and even collapse active areas throughout processing as the reconstruction filter does. Contrary to that, the reconstruction filter rather inflates them, as is already clearly observable in Figure 5.1. Moreover, for many types of local projection operations, active areas do not necessarily exist on the initial level $m = 0$ until the transform to the first level is completed. In such cases, active areas in any dimension only get generated after filtering in this dimension is finished. If, in turn, active areas may only be generated after the filtering in a respective dimension, and thus inflate the support area to be processed in the next step, it might be worthwhile to take care of the order of filter operations in order to additionally control the overall computational cost—a phenomenon that is hard to get across in words but a lot easier with the following example.

Figure 5.7 illustrates the impact of a chosen order of individual filtering steps in each dimension that are needed to complete the transformation to the next level. The reddish, axis-aligned line is being transformed to level $m = 1$ using a local projection operator. The optimal order shown in the top row first completes the y-direction which only involves one single line of data. The more costly order given in the bottom row generates two active areas by first processing the x-direction. As a consequence, the second step, the processing of the y-direction, involves a whole set of lines inside the active areas.

![Diagram](image.png)

**FIGURE 5.7** Filtering steps for a local projection operation of a single axis-aligned line in 2D, from approximation level 0 to 1, at location $(x_0)$. Top row: Optimal order of filtering. Bottom row: More costly order of filter steps.
5.3.2 4-dimensional Case

During the oracle mode, the system's core may decide to insert new image data into the eMRA representation, according to the rules given in Sections 4.7 and 4.8. It is then a difference image that needs to be incrementally inserted into the hierarchy. The difference image represents a single, axis-aligned $zt$-plane, which is written into and processed in the additional buffer.

The projecting of a complete $zt$-plane again requires global filtering in these two dimensions. Hence, active areas cannot be identified but in the $x$ and $y$ dimensions, which are centered on the sample location $(x_0, y_0)$ corresponding to the COP of the new image data. Again, the general processing scheme for a local projection operation may thus be readapted for the 4-dimensional case.

1. Set current level to $m = 0$;
2. Identify active areas in $x$ and $y$, according to the position of $(x_0, y_0)$ on the current level $m$;
3. Successively apply projection filter in $x$- and $y$-direction locally and in $z$- and $t$-direction globally;
4. Set new current level to $m = m+1$;
5. If $(m <> M)$ go back to step (2);

Figure 5.8 shows this projection procedure for the first filtering steps from approximation level $m = 0$ to level $m = 1$ for the location $(x_0, y_0)$. The illustration again visualizes the inflation of active areas due to the projection filter. Moreover, the support areas in the $x$- and the $y$-direction do not exist before filtering in the respective dimension is completed.

FIGURE 5.8  Filtering steps for a local projection operation of a single $zt$-plane in 4D, from approximation level 0 to 1, at location $(x_0, y_0)$. Top row: Optimal order of filtering. Bottom row: More costly order of filter steps.
Clearly, there is more than just one possibility of how to string together the individual filtering steps that are necessary to complete one level. The top row of Figure 5.8 shows the optimal order whereas the bottom row shows a correct yet far more costly sequence. It is obvious that after the \( x \)– and \( y \)–directions are completed, many \( zt \)–planes instead of just one need to be processed. In terms of the computational costs, it is therefore crucial to first find and then follow an optimal order.

Note that reordering of individual filtering steps at any level of the hierarchical representation is allowed as long as each required step is fully completed.

### 5.3.3 Buffer Clearing Strategy

The general processing scheme for local projection operations as is listed earlier in this Section 5.3 functions well for any local projection from approximation level \( m = 0 \) to \( m = 1 \). For any \( m > 1 \), however, the scheme is still valid but needs to be extended by an additional clearing step in case just one additional work buffer is being utilized.

The fundamental problem that demands for such a corrective is the existence of intermediate results from any filtering operation which are, of course, also stored in the additional work buffer. Note that, although they do potentially occur in all frequency subbands, it is only the intermediate results stored in the lowest–frequency subband on any level of approximation causing the problem—recall that the low–frequency subband is the one further subsampled during the transform to the next level.

Due to the stepwise magnified support areas from one approximation level to the next on the projection path it might happen that the filtering process in any direction wrongly involves such outdated intermediate results from a previous level. Of course, the deeper the hierarchy, the more likely such an unintentional situation gets because active areas become large relative to the subband's extents. Note that the problem arises independent of the dimensionality of the data field.

Further note that an analogue situation does not occur on the reconstruction path, basically because of the active areas' converse shift in size. That is, the support areas get scaled down from level to level during the back–transform in contrast to the projection path. As a consequence, the algorithm executing a local reconstruction only reads from inside valid areas that have been written directly beforehand—which is not the case on the projection path, as explained above.

A simple solution disregarding any resource concerns might allocate a second additional buffer. The two buffers may be alternately used as read and write buffers which offers the possibility to completely clear the buffer that was read from before switching the buffer's function.

It is, however, feasible to still get by with a single additional buffer. The following strategy is integrated in every local projection process the system performs. The low–frequency subband gets cleared only very selectively after the completion of a filter operation in any direction on any level, with the goal to erase intermediate results that are (1) outdated and already processed, (2) not completely overwritten by any new intermediate result and that (3) may potentially cause problems due to their position. As a matter of course this selective clear operation must imperatively preserve all valid results from filter operations just completed.

Following such a selective buffer clearing strategy requires to detect and handle many different cases. In a 2–dimensional setting, for instance, already 14 different cases may be
identified, with three of them getting solved automatically. The respective clear areas are determined and erased immediately after completion of filtering in any direction.

An interesting side note about a compilation issue may be added at this point. Extensive optimization of the program code implementing the local projection operation, which is done by the compiler's backend modules, fails as soon as the selective clearing code is integrated. The compiler, however, erases parts of the clearing code from the binary file because preconditions that allow to enter the respective code segments are being set after the code segment itself. All of this code additionally runs in various tricky nested loops.

The compiler employed, however, is part of the MIPSpro compilers family, version 7.3.1.2m. The extensive optimization that fails in the above described situation is referred to as the 2nd-level optimization which is the compiler's default setting as soon as optimization is turned on. The 1st-level optimization, however, performs a local optimization strategy and outputs binary code with a remarkable speed-up, too.

5.4 COMPUTATIONAL PERFORMANCE

The primary motivation for the construction of local operators is the need for finding a way to efficiently perform (1) data readouts for rendering and (2) update operations to gradually and incrementally refine the eMRA during the oracle mode.

The local operators as introduced in this Chapter 5 demonstrate a possibility to compute partial reconstructions and incremental updates of wavelet-transformed data without requiring computationally costly back- and re-transforms. Therefore, after having discussed the basic concept and their functioning, the achievable benefits in terms of computation time need to be analyzed.

Before commenting on the measured performance of both the local reconstruction and the local projection operator in Section 5.4.2, the importance of finding an optimal sequence of filtering steps for the transformation to the first level is again emphasized by giving a brief analysis.

5.4.1 First-level Transform

The optimal sequence of filter operations may greatly support the efficiency of the local operators—or, vice versa, not following the optimal ordering may seriously lessen the computational advantage achieved with the local operators. Commented examples were given in Figures 5.5, 5.7 and 5.8.

In spite of the fact that an optimally arranged filtering sequence only affects the transformation costs to the first level—active areas do exist in all dimensions once the transformation to level \( m = 1 \) has been completed—it significantly influences the overall computational costs. The reason for this impact is the costs of the first-level transform dominating the overall costs in higher-dimensional problem settings, as is pointed out below.

A virtual cost of \( c_{1D} \) shall be assigned to the complexity of filtering a data field in 1D. In this case, the total cost for the transformation of a squared 2-dimensional data field to level \( m = 1 \) is given by \( 2 \cdot c_{1D}^2 \)—twice the same amount because of the separation of the filter operation in any dimension. Due to subsampling, the projection to level \( m = 2 \)
5.4 Computational Performance

causes an additional effort of \(2 \cdot (c_{1D}/2)^2\). Proceeding to the maximum approximation level \(M\) and summing up yields the geometric series given in Equation 5.6.

\[
2c_{1D}^2 + 2 \cdot \left(\frac{c_{1D}^2}{4}\right) + 2 \cdot \left(\frac{c_{1D}^2}{16}\right) + \ldots = 2c_{1D}^2 \sum_{m=0}^{M} \frac{1}{(2^m)}
\]  

(5.6)

The value of the above sum can be specified for any \(m \geq 0\) using the properties of geometric series, as is shown below for an \(n\)–dimensional setting.

\[
n c_{1D}^n \sum_{m=0}^{M} \frac{1}{(2^n m)} = nc_{1D}^n \left(\frac{1 - q^{m+1}}{1 - q}\right) \quad \text{with} \quad q = \frac{1}{2^n}
\]

\[
\rightarrow nc_{1D}^n \left(\frac{1}{1 - q}\right) \quad \text{for} \quad M \to \infty \quad \text{and} \quad |q| < 1
\]

(5.7)

Table 5.1 below confronts these costs of a transformation to the first and the third level of approximation, dependent on different dimensions from 1D to 4D. Additionally, the accumulated costs for a level–3 transform are given, too. In a 1–dimensional problem setting, the transformation to the first level is already half of the cost of all following steps if \(m \to \infty\). In 2D, the costs to the first level even dominate the overall complexity. In 3– and higher–dimensional settings, this bias gets even more drastic. Moreover, the difference between a transform to an intermediate approximation level and the maximum possible depth becomes negligible.

As a consequence, the higher the dimensionality, the more may be gained by handling the first–level transform as efficiently as possible. Note that the statements made for the projection path are also valid for the reconstruction path.

### Table 5.1

<table>
<thead>
<tr>
<th>Dimensionality</th>
<th>To level 1 : (m = 0)</th>
<th>To level 3 : (m = 2)</th>
<th>For (m \to \infty)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–dimensional: (n = 1)</td>
<td>(c_{1D})</td>
<td>(\frac{7}{4}c_{1D} = 1.75 \cdot c_{1D})</td>
<td>(2 \cdot c_{1D})</td>
</tr>
<tr>
<td>2–dimensional: (n = 2)</td>
<td>(2 \cdot c_{1D}^2)</td>
<td>(\frac{21}{8}c_{1D}^2 = 2.625 \cdot c_{1D}^2)</td>
<td>(\frac{8}{3}c_{1D}^2 = 2.667 \cdot c_{1D}^2)</td>
</tr>
<tr>
<td>3–dimensional: (n = 3)</td>
<td>(3 \cdot c_{1D}^3)</td>
<td>(\frac{219}{64}c_{1D}^3 = 3.42 \cdot c_{1D}^3)</td>
<td>(\frac{24}{7}c_{1D}^3 = 3.433 \cdot c_{1D}^3)</td>
</tr>
<tr>
<td>4–dimensional: (n = 4)</td>
<td>(4 \cdot c_{1D}^4)</td>
<td>(\frac{273}{64}c_{1D}^4 = 4.266 \cdot c_{1D}^4)</td>
<td>(\frac{64}{15}c_{1D}^4 = 4.267 \cdot c_{1D}^4)</td>
</tr>
</tbody>
</table>

5.4.2 Measured Performance

Since both local operators essentially work on 2–dimensional slices through the 4D data set instead of the 4–dimensional data field as a whole, they perform in the order of \(O(N^2)\) in contrast to \(O(N^3)\) in the latter case, as was already mentioned in Section 5.1, for instance. Again, the parameter \(N\) denotes the size of any dimension of some data set.
A more detailed analysis of the computational complexity, however, yields the total dependence shown in Expression 5.8, where \( N \) denotes the size of those dimensions that need to be filtered globally, in particular. These are, according to the previous sections, the \( z \) and \( t \) coordinates and match with the image resolution. The first term \( O(M) \) accounts for the depth of the transformation pyramid whereas the second term \( O(s_h) \) considers the size of the wavelet filter.

\[
O(M) \cdot O(s_h) \cdot O(N^2)
\]  

(5.8)

As was stated before, most of the computational effort needs to be done during the transformation to level \( m = 1 \). Thus, the overall complexity should show just a weak dependence of the first term in Expression 5.8, which is confirmed in Figure 5.9 below: The timing values reported in the left column for transformations to the maximal possible depth \( M \) only differ little from those in the right column for transforms to the first level only.

**FIGURE 5.9** Timings measured for different wavelet filters dependent on the resolution in \( x \) and \( y \), with \( z \) and \( t \) set to a value of 64.
Left column: Maximal depth of transformation.
Right column: Transformation to level \( m = 1 \).

These timings were measured on an IRIX-driven computer system with a single MIPS R12000 processor running at 400 MHz. The top row shows the values computed with
the Haar wavelet basis, the bottom row those for the Daubechies–6 basis. Each chart displays the time spent for computing a full wavelet transform of the complete 4-dimensional data set, for a local reconstruction as well as for a local projection operation, dependent on the resolution in \( x \) and \( y \), respectively. The size of the \( z \) and \( t \) dimension were both fixed at a value of 64.

The charts firstly illustrate an evident quadratic increase of the computation time spent for the complete transformation. Secondly, note that the curves of the local operators both do not exhibit a similar increase in computation time. In case of a wavelet transform to the maximal possible depth (left column of Figure 5.9) they behave linearly, which is reflected in the first term of Expression 5.8. In case of a level-1 transformation only, the curves of both local operators start to level off to near constant behavior at a certain point—as is emphasized even more evidently in the bottom chart of Figure 5.10.

![Figure 5.10](image)

**FIGURE 5.10** Timing charts comparing transformation times for complete decompositions and for local reconstructions, measured for different wavelet bases (\( z \) and \( t \) again set to 64).

This point can be described as the minimal size of \( x \) and \( y \) from which on the identification of active areas for local filtering, using some given wavelet, becomes possible—remember that in case of small data set sizes in any dimension, the identification of local
support areas might not be possible and global filtering in the respective dimension must be applied instead. However, once this position is arrived at, the computation time gets independent of the size of $x$ and $y$ as is stated in Expression 5.8.

Again, the bottom row of Figure 5.10 shows the values of $x$ and $y$ for which the computation time needed for the local reconstruction operation levels off to nearly constant behavior. The precise values, however, are dependent on the size of the respective wavelet filter as is indicated by the second term in Expression 5.8. The Daubechies-6 wavelet, for instance, starts showing a constant behavior earlier than the Daubechies-10 basis as a consequence of its smaller tap-size.

Measured data for the local projection operation exhibits the qualitative same characteristics as is the case for local reconstruction operations. However, note that the small difference in their computational performance between the local reconstruction and a local projection as is observable in all charts of Figure 5.9 is caused by the additional efforts needed for (1) the selective buffer clear operations and for (2) the local updates of the eMRA representation.

For comparison, the upper chart in Figure 5.10 shows the timings measured for a full decomposition, with the size of $z$ and $t$ again fixed at 64. The transform for each wavelet were completed up to the first level.

The results presented so far point out that the local operators as they are introduced in this Chapter 5 both exhibit the expected features—namely in terms of visual performance as well as in terms of computational performance. The timings given in the previous charts demonstrate the remarkable advantage regarding computational complexity of the concept of local operators in particular. In other words, they perform convincing from a qualitative point of view.

In terms of absolute computation times, however, a local reconstruction operation still consumes from somewhat less than one second using the Haar basis to more than seven seconds using the Daubechies-10 wavelet (Figure 5.10, bottom chart), despite of the considerable speed-up in comparison with the computing times needed for a complete transformation (Figure 5.9).

The switch-over from the initial IRIX-driven test platform to a Linux-based system running on a 2.8 GHz Intel Pentium4 processor significantly improves these numbers. Figure 5.11 contrast the timings for local reconstruction operations measured on the primal test platform with those achievable on the Linux system. As is to be expected, the qualitative characteristics stay the same whereas the time scale changes about a factor of 4.3. The performance of the Daubechies-4 or the Haar basis, for instance, allow for reconstructions, i.e. rendering, at interactive rates. However, note that all timing results were generated with the $z$- and $t$-dimension fixed to a value of 64 which matches a comparatively small image size.

For completeness, Figure 5.12 compares the two platforms with respect to the timings measured for full decompositions to the first level $m = 1$, revealing again the same qualitative run of the curves but with processing times scaled about a factor of 4.3, too.
5.4 COMPUTATIONAL PERFORMANCE

FIGURE 5.11 Comparison of timings for local reconstruction operations, measured on different computer systems. Top row: Timings measured on a MIPS R12000 processor @ 400 Mhz (IRIX system). Bottom row: Timings measured on an INTEL Pentium4 @ 2.8 GHz (Linux system).
FIGURE 5.12 Comparison of timings for complete wavelet transformations, measured on different computer systems.
Top row: Timings measured on a MIPS R12000 processor @ 400 Mhz (IRIX system).
Bottom row: Timings measured on a INTEL Pentium4 @ 2.8 GHz (Linux system).
CHAPTER

IMPLEMENTATION DETAILS

Aside from the underlying data representation (Chapter 4) and the set of local operators (Chapter 5) there are two additional yet more technical modules that make up important parts of the system. Firstly, Section 6.1 discusses supplementary information about aspects of the parameterization spaces that has not been mentioned up to now.

Furthermore, Section 6.2 presents the compression mechanism as a whole. Firstly, the needs and requirements a suitable compression has to fulfill are given in Section 6.2.1. Secondly, the basic principle the compression is built on is briefly recapitulated in Section 6.2.2, followed by the presentation of the custom-tailored in-core and file structures (Sections 6.2.3 and 6.2.4). Section 6.2.5 finally comments on the performance the presented mechanism is able to achieve both in terms of compression ratio and computational efficiency.

Note that Section 6.2.5 anticipates results concerning achievable compression factors from experiments based on real-world input imagery that will be presented in detail in Section 7.7.

6.1 PARAMETERIZATION

Several facets of the parameterizations employed have been introduced earlier, as is summarized below.

- **Light slab:**
  The setting employed in both Light field rendering [75] and the Lumigraph [45], the so-called light slab, also referred to as the standard 2-plane parameterization, is introduced in Section 2.2.2, along with the basic principle of 3-dimensional ray parameterization using four parameters in order to capture a subset of the Plenoptic function [2].
State of the art:
A compilation of available parameterization techniques used in the context of IBR is given in Section 2.3.2. Moreover, a brief discussion about the basic alternatives for the parameterization of image data and for ray parameterization in particular is included.

System component:
Section 3.2 presents the two parameterization settings readily available in the corresponding system module. In particular, the basic design of both of them along with some of their pros and cons are given.

Ray-based vs. slicing:
The motivation for sliced edit operations together with the major differences to the ray-based approach are introduced in Section 5.1.5.

The remainder of this Section 6.1 supplements the above information with several aspects not discussed so far.

6.1.1 Cylinder-plane Setting
The configuration of the cylinder-plane parameterization has already been given in Section 3.2. Figure 6.1 below shows a more detailed illustration of the setting with the black arrow representing an example ray.

The parameterization uses the four parameters of a viewing ray intersecting with a cylinder and a plane. The cylinder circumscribes the object of interest centered at the origin. The plane, on the other hand, is centered inside the cylinder, aligned parallel to the cylinder's axis and orientated perpendicular to the current viewing direction as is shown in Figure 6.1.

This setting has its advantages, namely the full horizontal coverage and the absence of artifacts caused by the disparity problem, as was stated in Section 3.2. Moreover, in [82] it is reported that the cylinder-plane parameterization proved to be considerably better and generally more homogeneous in terms of SNR as compared to the standard 2-plane parameterization. This, however, applies to some of the important camera paths in the given setting, i.e. horizontally circular paths. Naturally, a cylindrical or spherical setting is better suited for such circular camera motions as is the standard 2-plane parameterization for planar camera motions.

![Cylinder-plane parameterization](image)

**FIGURE 6.1** Cylinder-plane parameterization.
However, the cylinder–plane setting also shows disadvantages. Beside the slightly more expensive ray intersection calculations due to the angle $\phi$ involved as compared to the standard 2–plane parameterization it is mainly the vertically fixed alignment of the plane that causes the following problem.

If the viewpoint for reconstructed views is moved off the horizontal plane at $z = 0$ while still holding the optical axis towards the origin, i.e. the object's center, foreshortening effects will occur due to the perspective distortion of the image plane on the parameterization plane, i.e. the $st$–plane.

It is, however, important to note that such effects do only occur for operations based on slicing or if ray–based and slicing are intermixed. That is, for purely ray–based operations, i.e. data insertion and/or extraction, foreshortening will never be an issue since the perspective is automatically taken into account when computing ray intersections. If, on the other hand, any new view gets reconstructed from a ray database by using an operator based on slicing the perspective distortion will be clearly visible, as is shown in Figure 6.2.

As a consequence, the cylinder–plane parameterization is not very well suited for clearly elevated positions above the $z = 0$ plane in combination with the operators based on slicing—unless not corrected for. An adjustment of the perspective distortion might be accomplished by post–warping views reconstructed using slice extraction, and pre–warping new input images prior to insertion. Such additional operations may be done using the texture hardware of a graphics subsystem for image resampling, for instance.

The foreshortening effects may, however, be completely avoided if the restrictions on the plane’s freedom of movement are reduced—which is the case for the sphere–plane parameterization as is restated in Section 6.1.2 below.

**FIGURE 6.2** Foreshortening occurring with the cylinder–plane parameterization.

a) Horizontal view at $z = 0$.

b) Perspective distortion due to an elevated camera position.

### 6.1.2 Sphere–plane Setting

The basic configuration of the sphere–plane setting has also been introduced in Section 3.2. However, Figure 6.3 gives a more detailed illustration of the design. Again, the black arrow represents an example ray.

The plane centered inside the sphere is always oriented perpendicular to the current viewing direction, which avoids the perspective distortion effects implicitly, also in case of operations based on slicing, of course.
The sphere–plane parameterization shares many analogies with the DPP [16] as was already mentioned in Section 3.2. Among the known light field parameterizations, the DPP introduces the least number of biases concerning view dependencies. As is derived in [16] it is the planar surface orthogonal to the viewing line’s direction that introduces the fewest correction factors, while the surface for the first intersection point may be any surface—properties the sphere–plane setting shares with the DPP.

Again, the major difference is given by the DPP’s characteristic to explicitly generate uniformly sampled light field representations as was already stated in Section 3.2.

6.1.3 Ray–based vs. Sliced Edit Operations

The motivation for the usage of sliced edit operations was formerly given in Section 5.1.5 on page 91. There are, however, the following implications and issues that have not been discussed so far.

Identification of slices. The individual slices are still addressed using the parameterization in use. That is, the ray coinciding with the optical axis of a given input image or a given view for reconstruction is intersected with the sphere, or the cylinder, respectively, yielding a pair of \(x\)– and \(y\)–coordinates. The corresponding slice in 4D is accessed by fully traversing the \(z\)– and \(t\)–coordinates for the given \(x\) and \(y\) values.

Note that \(x\) and \(y\) correspond to the first two intersection parameters \((\varphi, \theta)\) of the cylinder–plane parameterization or \((\varphi, \psi)\) of the sphere–plane parameterization, respectively, as was introduced in Section 5.1.5 for ease of notation. Likewise, the remaining two intersection parameters on the \(st\)–plane are referred to as \(z\) and \(t\).

Filtering according to basis functions. The above mentioned intersection computations will most probably not exactly hit a grid point in the \(xy\)–parameter space. As a consequence, it will not be possible to clearly associate a single slice through the 4-dimensional data set with the exact intersection point. The solution to this problem involves the surrounding grid points in the \(xy\)–parameter space and their assigned slices. That is, the four slices of the four surrounding grid points of an intersection point get weighted according to the exact position of the intersection point, thereby taking the quadra–linear basis functions into account. For a reconstruction operation, the four slices are blended together, according to their weights, by using accumulation buffer support of the graphics sub-system. For an update operation, in turn, the difference image representing the new infor-
6.2 COMPRESSION

Information gets split into four weighted fractions corresponding to the four neighboring grid points of the exact intersection point.

Consequently, it is important to note that—due to the quadra-linear basis being employed—a local reconstruction as well as a local update does not involve one but rather four slices in their computation in case of sliced edit operations on the transformed data.

**Camera's freedom of movement.** Since every slice through the 4-dimensional data set is associated with its corresponding pair of x- and y-coordinates the camera is locked onto the surface the xy-parameter space is defined on, for all sliced edit operations. That is, in case of the cylinder-plane parameterization, a cylindrical shape and in case of the sphere-plane setting the spherical surface.

While this is a grave limitation on the rendering side, it is not during data acquisition if a device is being used as it will be described in Section 7.1. The video camera mounted on the object rig employed is conceptually able to cover a sphere around an object of interest which is exactly supported by the sphere-plane parameterization. As a matter of course, this limitation imposes serious restrictions also during data acquisition in case a hand-held video camera for free image acquisition will be used.

It is important to note that the above limitation is not present in case of any ray-based operation. For sliced edit operations, however, the restriction may be weakened in a limited way, again through pre- and/or post-warping operations, respectively. An input image shot by a camera off the spherical or cylindrical surface in radial direction, for instance, may be corrected by a simple zoom operation.

6.2 COMPRESSION

The compression mechanism developed in the course of this thesis may be characterized as a 3-stage compression scheme. The first and most important part is represented by the in-core compression, the second part by the file structure used to permanently store an eMRA representation on disk and finally by an additional compression stage applied to the written file.

Section 6.2.3 introduces the structure used to store and access the compressed in-core data. Section 6.2.4 describes the file structure used to write data to disk as well as the additional compression applied to it. Before discussing all of these three compression components, Section 6.2.1 lists the requirements a suitable compression mechanism has to satisfy, followed by a brief summary of the compression principle applied (Section 6.2.2).

6.2.1 Requirements

The design of an in-core and out-of-core compression for light field data, i.e. the eMRA representation in particular, is guided by several constraints and goals. Some of them have already been mentioned and discussed in earlier publications. In [75], for instance, some of the characteristics of a general light field compression are discussed, including (1) the present data redundancy that needs to be removed as completely as possible, (2) the necessary random access that must be guaranteed at any point in time, (3) the need for a computationally low-cost solution on the decoding side, possibly without the need for support by special purpose hardware, and finally (4) the insight, that an asymmetric compression scheme may be sufficient in the context of LFR since data is assembled and compressed prior to viewing.
Moreover, two of the prerequisites for the design of a compression scheme for light field data based on wavelets in particular are given in [98]. Firstly, the random access to arbitrary data items which needs to be supported by a compression module is mentioned as well. Secondly, the demand for computationally efficient decoding operations is extended to real-time reconstructions, possibly supported by a caching mechanism.

However, the following catalog briefly summarizes the important requirements in the present setting, not only for an in-core but also for an accessory out-of-core compression scheme. Clearly, any compression mechanism aims for the optimal compression factors in a given setting and hence, the best possible compression ratios do not need to be listed as a requirement.

- **Random access:**
  Every data item stored in a compressed representation must be randomly accessible at any time, as is stated as being one of the essential requirements in all papers dealing with compression of LF-like data. Note that this property is not guaranteed at all by all compression techniques. Compressed multi-media based data streams, for instance, are often encoded using a predictor-corrector model, e.g. reference frames and subsequent incremental correction and update frames in case of MPEG-encoded video data, which does not allow to directly access an individual frame.

  Since, in the present case, data will most likely not be accessed in a predictable or even sequential order—a user may request a reconstructed view from any possible location in the parameter space—random access to any compressed data item must be provided.

- **Computational efficiency:**
  Aiming at JIT rendering (Section 5.2.3 on page 96) the additional computational overhead needed for the handling of a compression scheme must not become extensive. Moreover, although the demand for efficient rendering solely affects the decoding of data, the encoding of the representation, on the other hand, also needs to be completed in a moderate amount of time, due to the following reason.

  Since the transformation to wavelet space is part of the construction of the eMRA representation during the approximation mode, a wavelet-based compression is not possible but right after the completion of this first mode. That is, a wavelet-based compression is to be established after the approximation mode but preferably before the oracle mode, i.e. after the interpolated estimate is available but before the incremental update operations start. As a consequence, the compression scheme employed may be asymmetric, but not as computationally expensive as the encoding stage of a vector quantization [75] for instance.

- **Storage efficiency:**
  Naturally, addressing the best possible compression ratio simultaneously means minimizing the storage needs for a compressed representation. The present setting, however, imposes an additional requirement concerning the maximal storage needs, as is pointed out below.

  After the projection of an initial small set of input images, the light field representation might still exhibit undefined regions, so-called NO_DATA areas (Section 4.6). The task of the HSDI procedure performed as part of the approximation mode is to close such gaps. On the one hand, an initial light field representation with its undefined areas may already benefit from a compression scheme that is able to handle and
store existing data only. On the other hand, the HSDI procedure’s goal is to fill in the
missing data, i.e. it generates data which fills up the structure storing the data. As a
consequence, if the data structure implementing a compression scheme is used during
both mode’s of the core it must be guaranteed that it does not consume significantly
more memory than a linear array does—even in its fully populated state.

♦ Support for dynamic data:
A suitable compression scheme must additionally offer support for dynamic changes
of the data stored. Since the system’s core not only computes reconstructions but also
accepts local incremental updates during its oracle mode support for dynamic data is
an obvious but difficult requirement.

Apart from the best possible compression factors in the given setting, all above require¬
ments rather apply to the in—core compression than to the compressed data permanently
stored on disk. There is, however, an additional desirable property to be added affecting
the file structure.

♦ Progressivity:
When reading back to main memory a compressed eMRA representation that was
previously transferred to disk—potentially over a network link—, it may be eligible
to be able to start reconstructing views, i.e. rendering, while the loading process is still
ongoing. Hence, progressivity may be added to the list of requirements concerning
the file format.

The following Section 6.2.2 addresses the basic principle of the compression scheme
employed before the implementation of the in—core and the file structure that satisfy all
of the previously mentioned requirements are being introduced.

6.2.2 Compression Principle

The wavelet transform was introduced to the setting given in this thesis mainly due to the
following three reasons. Firstly, the multiresolution properties of the wavelet space maybe
matched with the hierarchical representation established during a necessary HSDI pro¬
cess, resulting in the eMRA representation introduced in Chapter 4. Secondly, the local¬
ization properties of wavelet bases enable the design of only locally working operators as
they were introduced in Chapter 5. The third reason finally is compression since the
wavelet transform offers a well—known potential for high compression gains.

Wavelet—based compression, however, is a transform compression method as was
anticipated in Section 2.3.3 and deepened in Section 4.3.6 on page 64. In the latter sec¬
tion the principle of coefficient thresholding, leading to an efficient and controllable lossy
compression was discussed in detail. Following this principle, a data structure implement¬
ing the compression scheme must primarily be able to only store the non—discarded coeffi¬
cients. As a matter of course, an additional bookkeeping mechanism in order to keep
track of present and neglected coefficients becomes necessary.

The data structure implementing such a selective storage container with the required
accurate bookkeeping designed for the in—core compression is introduced in the following
Section 6.2.3. Moreover, it discusses alternatives to the solution finally chosen and gives
the motivation for the custom design.

It is, however, important to note that the scheme as is implemented and used in the
context of this thesis does not perform any requantization of wavelet coefficients which is
usually done and enables even more effective compression ratios. The reason for this is
given by the highly dynamic environment mentioned in the previous list of requirements. Requantization is, however, also not part of the compressed file format, as is deepened in Sections 6.2.4 and 6.2.5.

6.2.3 In-core Data Structure

The literature offers many different alternatives to efficiently store wavelet-compressed data of any kind. Probably one of the best known representations at all is the Zerotree of wavelet coefficients [113], one of the first model of the huge class of tree-like wavelet representations. Its improved successor, the SPHIT model [106], is another well-known example of this class.

Note that the majority of the approaches that utilize the wavelet transform in the context of LFR (Section 2.3-4) also employ a tree-like scheme for data compression, thereby taking advantage of the hierarchy of the wavelet representation in a quite natural way. Such trees of coefficients need to be serialized in an appropriate way for the purpose of streamed transmission and/or permanent storage in a progressive format. Although handling multi-dimensional data, it often remains unclear how multi-channel data, i.e. data items with several components, e.g. aRGB, fits into the scheme.

However, the concept of the zerotree and any tree built upon the remaining wavelet coefficients after thresholding either fails in a dynamic setting or forfeits its optimal properties since its fundament continually changes. Consequently, the design of the in-core compression scheme is not guided by the zerotree or an alike scheme.

An alternate representation for the efficient storage of the remaining wavelet coefficients may be found in the context of sparse matrices. The well-known Harwell-Boeing format [37], for instance, is explicitly designed to store the non-zero entries of a sparsely populated matrix using a very small memory footprint. It manages the non-zero entries in a contiguous linear array which may be complemented with additional structures allowing direct access to individual entries [7]. Even though the basic mechanism of such a representation may conceptually be extended to multi-dimensional spaces it is again not a suitable solution for the given setting because the Harwell-Boeing format does not allow any modification of the matrix after its structure has been fixed, i.e. it is entirely improper for a dynamic environment.

However, the set of remaining coefficients after the thresholding procedure may be conceived as an abstract heap of data items with an associated 4-dimensional index to each of which. These non-ambiguous indices may be interpreted as a kind of key uniquely identifying the respective coefficient, leading to the idea of a hash table [109]. Although originally designed for static data, fully self-organizing implementations of a hash table are readily available, e.g. the hash_map container which is part of the STL [95, 122]. However, a hash table works most efficiently for the number of entries it was initially configured and set up. Experiments with the above mentioned STL container showed that it is, in principle, possible to store the coefficients and their index values in such a structure, even with a dynamically changing amount of data.

Nonetheless, a self-organizing hash table is far from being a suitable data structure in the present setting, too. Firstly, operations performing restructuring tasks such as a container enlargement are extremely costly. Secondly and even more importantly, hash tables represent a class of data structures built for fast access to data items with $O(1)$ average time complexity for insertion and removal operations [110]. They are by no means
designed as a container for data compression. Hash maps and comparable data structures may cause a considerable additional memory overhead, especially if they are self-adapting. That is, the memory needs for internal container management is extensive and may even exceed the effective amount of user data.

Nevertheless, the basic idea of using the index values as keys to the actual data items stored in a collection of buckets led to the design of a custom data structure implementing the in-core compression mechanism. Figure 6.4 gives a schematic overview of this structure that is introduced below.

**Basic setup.** Access to individual data items is enabled with the help of a primary container element, that is implemented as a dynamic array holding references to a collection of buckets. Each of these buckets again holds two dynamic arrays the first of which stores the actual multi-channel data items whereas the second array stores the corresponding index values—which is necessary in order to keep track of available and discarded coefficients. These two arrays are synchronized, that is if an index value searched for is found at the position $\text{indices}[i]$, its corresponding data is to be read from $\text{values}[i]$. Each bucket keeps track of the actual number of filed index–value–pairs by adequately updating the $\text{nof_values}$ variable.

Before addressing individual access operations for data insertion and data removal, for instance, the effective structure declarations of a bucket as well as of the container itself are given, using valid C/C++ notation.
struct sLFO_Compr_Bucket {
    float * values;
    unsigned int * indices;
    unsigned short nof_values;
    unsigned char flags; //bit 0=full, bit 1=unsorted
};

struct sLFO_Compr_Container {
    sLFO_Compr_Bucket ** buckets;
    unsigned short nof_buckets;
};

Initialization. Except for a reference pointing to the container the whole structure is empty in its initial state. The container itself gets allocated once all necessary parameters have been specified and made available to the structure. These are firstly the resolution of the light field in all dimensions—which specifies the maximum capacity of the data structure—and secondly the capacity of every single bucket, the so-called bucket size. Having this information ready, the number of buckets needed gets determinable and is stored using the variable nof_buckets in the container structure. Individual buckets are not allocated and initialized before actually used, i.e. not before the first time accessed. Index values are linearly mapped to the virtual collection of buckets, i.e. the proper bucket for a given index value is determined by a single integer division of the index by the bucket size.

Data insertion. In principle, a write operation first determines the affected bucket and inserts the index–value–pair according to the bucket’s state which is stored using the flag field of the bucket structure. Bit one, for instance, indicates whether the arrays in the respective bucket are sorted or not. In a sorted state, the index–value–pairs are ordered by ascending index values, resulting in faster access times due to a binary search but eventually causing some shifting overhead during data insertion. In an unsorted state, on the other hand, data insertion gets reduced to a simple append operation while data access becomes slower due to a linear search. Clearly, the nof_values variable needs to be updated after every insertion operation. Note that the arrays of a bucket are initially small. When filled up due to repeated insertion operations they get dynamically expanded, if necessary up to the maximum capacity specified. Consequently, the storage capabilities of the respective bucket gets extended at the cost of increased memory consumption, of course.

Data read out. Once the affected bucket is determined via the single integer division the queried index value is looked for in the indices array, which is done using a fast binary search in case the bucket is sorted or by linearly searching for it otherwise, as was mentioned before. If the index value is not found, the corresponding data item, i.e. the coefficient, is not present. If found, the corresponding data is read from the values array and returned.

Data removal. An index–value–pair to be deleted is accessed in the very same way as it is done for data read out. If found, it is removed from the bucket which then needs to be adapted. Clearly, the variable nof_values is decreased. The removal of the data itself additionally causes shifting of remaining data in order to avoid gaps. If possible, the arrays are compacted in order to free up memory not used anymore. Moreover, empty buckets are completely removed. In this case, the respective reference in the container structure needs to be adapted accordingly, of course.
Reorganization. It is, however, necessary to pay an additional overhead for reorganization of individual buckets in certain situations as is pointed out below. Individual bits of the flags field which is a member of the bucket structure are used to keep track of these situations and the buckets' state.

1) Sorted/Unsorted arrays:
An initially unsorted state enables faster data insertion, as was mentioned before, whereas sorted arrays allow for more efficient read operations. This property is exploited in order to support the core's modes, i.e. buckets are left unsorted during the approximation mode with its numerous write operations due to data projection and interpolation. When switching to the oracle mode, the buckets' data arrays are sorted by and by in order to optimize access times for data reconstruction, i.e. rendering.

2) Capacity used:
An even more important reorganization is unavoidable in case a bucket gets populated such that the storage of the data and index values together consumes more memory as a linear array configured for all data items would. As soon as this situation occurs, the values array of the concerned bucket is expanded to its maximum size, i.e. the specified bucket size, and the indices array is dropped, returning the thus freed memory. It is important to note that the array storing the indices is in fact not required anymore in case the values array is extended to its maximum size, since the data array is prepared to store all values associated with the respective bucket. Hence, data items are directly accessible using the index value. To be more precisely, the affected bucket is identified as was done before by computing

\[
\text{index_value} / \text{bucket_size}; \quad //\text{integer division}
\]

whereas the index into the corresponding values array is given by

\[
\text{index_value} \% \text{bucket_size}; \quad //\text{modulo operation}
\]

Note that the data structure as a whole will never consume significantly more memory than a single contiguous array for all data items does—even if all buckets are reorganized in the above manner. The difference is limited to the container structure providing all references to individual buckets. Further note that, if thus reorganized, access to a specific data item is as efficient as using a single array, with just one additional lookup in the container structure. Consequently, the data structure features a nice trade-off between access speed and compactness—the more memory needed, the faster data access becomes, i.e. the more filled with data, the more the structure behaves like a linear array of the same capacity, namely in terms of access time as well as memory consumption. On the other hand, data access gets more complex and thus more expensive only if less data is stored which allows for a compact representation, yielding a quite natural behavior.

The in-core data structure as is described above satisfies all requirements previously demanded in Section 6.2.1. Moreover, all algorithms and procedures previously introduced, namely the construction of the eMRA, including the HSDI and the projection to difference space, as well as the set of local operators, do run on top of the compressed structure without any restrictions.

However, there are a few more characteristics worth being mentioned, namely the issues briefly discussed subsequently.

- Again, in-core compression is achieved by way of only storing the set of remaining coefficients after thresholding. The compression scheme introduced offers mecha-
nisms (1) to store this set in a compact manner and (2) to freely access and change each data item. Due to the highly dynamic setting, no kind of requantization of coefficients is performed.

- The choice of the bucket size that needs to be known for the initialization of the container structure poses another trade-off to be considered. A small value means faster data access into the buckets’ arrays, especially in an unsorted state. On the downside, the management overhead increases since a small bucket size automatically means that more buckets are necessary for an equal capacity—and vice versa for the choice of a large bucket size. However, a value of 256 proved to be an appropriate compromise in the given setting.

### 6.2.4 File Structure

A progressive file format offers the nice possibility to start rendering from an incompletely loaded data set. That is, as soon as the lowest-frequency subband is read back to main memory reasonable yet not very detailed reconstructions may be computed. Any of the higher-level subbands will add more detail information to the representation. Note that this is a well-known general principle applicable to any kind of wavelet-transformed data which has in fact been used in many different applications.

To have the coefficients from the lowest-frequency subband at a file’s opening, followed by the other subbands on a level-by-level basis, requires a suitable reordering of subbands and coefficients such that a serialized stream can be written to disk and also read back again.

The custom file format divides written data into three parts, which are (1) the header, (2) a bitfield with important control data and finally (3) the light field data itself. Figure 6.5 gives an overview of these three parts. The first portion, i.e. the header, stores all necessary information needed to precisely describe the light field data in the subsequent third part. This last part includes all non-neglected coefficients of all subbands in a fixed order. Of course, the series of coefficients from the lowest-frequency subband are written foremost, followed by the coefficients of all $M \cdot 15$ subbands. Expression 6.1 below gives the total number of subbands of wavelet-transformed data dependent on the dimensionality denoted by $\text{dim}$, yielding an amount of $1 + M \cdot 15$ for a transform to the maximum level $M$ in 4D.

$$1 + \sum_{m=1}^{M} (2^{\text{dim}} - 1) = 1 + M \cdot (2^{\text{dim}} - 1)$$

Note that part three only includes the multi-channel data stored in the values array in the buckets of the in-core data structure (Section 6.2.3), i.e. no index data is included. Consequently, some other mechanism is necessary in order to keep track of available and disregarded coefficients. For this purpose, the second part of the file format stores a bitfield which indicates the presence or absence of every single coefficient using one bit for each.

A bitfield synchronized with the sequence of coefficients itself results in an even more compact representation, at the expense of random access. Note that a combination of the container and buckets structure presented in the previous Section 6.2.3 without the indices arrays but a global bitfield structure instead does not work since the information about the association of stored coefficients with their index value would be lost.
Any written file using the described format is additionally compressed using one of the readily available implementations of a loss-less Lempel-Ziv encoder [147], e.g. gzip and gunzip or compress and uncompress on UNIX systems.

However, with the focus on as aggressive compression as possible regarding the size of compressed files one would typically use a lossy requantizer instead of a loss-less second stage, of course. With the main interest of the compression module being put on the dynamic in-core data and all algorithms interacting with it, a Lempel-Ziv encoder is merely chosen due to its simplicity and availability.

Read back of thus compressed files runs as a background light-weight process, thereby enabling reconstructions once the lowest-frequency subband is available. The implementation of the container-buckets data structure which is not thread-safe in its current version requires the usage of a transfer buffer in order to avoid concurrency problems such as excessive locking, for instance [46].

<table>
<thead>
<tr>
<th>file structure</th>
<th>part (1)</th>
<th>part (2)</th>
<th>part (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>part (1): header</td>
<td>file name, parameterization, etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>part (2): field of bits</td>
<td>00100010011000010010101011000..</td>
<td></td>
<td></td>
</tr>
<tr>
<td>part (3): coefficients</td>
<td>coefficients $c_m$</td>
<td>coefficients $d'_m$</td>
<td>coefficients $d'^{m-1}$</td>
</tr>
<tr>
<td></td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td>$\ldots$</td>
<td>coefficients $d'_m$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$i = 1..15$, $0 &lt; m &lt; M - 1$</td>
<td>coefficients $d_0$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 6.5** Schematic illustration of the custom file format allowing for progressive read back.

### 6.2.5 Compression Performance

Two different sets of key parameters are important in order to evaluate the effectiveness of the in-core and the file structure. The computational performance of the system utilizing the in-core compression scheme is a first indicator, especially in comparison with the uncompressed setting.

Clearly, the even more important parameters are the compression ratios achievable, separately measured for the three stages of the overall compression scheme, i.e. the in-core structure, the file format and its additionally compressed version. Both of these two indicators are given and discussed below.
Computational performance. Table 6.1 gives the average decelerating factors of important algorithms when run using the compressed in-core representation. These parameters have been determined on a Linux-driven computer system with a single Intel Pentium4 processor running at 2.8 GHz and using the Haar wavelet basis, mainly because of its lower computational costs due to its highly compact support. The reference values have been set by running the same algorithms on the same computer system in an uncompressed state, of course.

Note that thresholding the coefficients is part of the transform to wavelet space. Consequently, beside numerous write operations during the transformation itself, thresholding additionally causes a certain number of data removals. The locally working algorithms tested—a mix of local update and local reconstruction operations—cause significantly less management overhead inside the in-core data structure which is reflected by the difference between the factor of the global transform and the one of the local operators given in Table 6.1. It is, however, important to note that none of the given numbers is influenced by the computationally intensive task of finding the corresponding threshold for a given amount of signal energy to be preserved, i.e. the determination of the threshold was factored out during the above mentioned test runs.

Table 6.1 Compression performance in terms of computational costs.

<table>
<thead>
<tr>
<th>Class of operations</th>
<th>Decelerate factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global wavelet decomposition</td>
<td>1.9</td>
</tr>
<tr>
<td>Local operators</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Experiments with different compilers on the same hardware revealed that binaries written by the Intel ICC 7.0 compiler do run faster than the same code translated by the gcc compiler, version 2.9.6. The difference lies in the order of 5% and comes at no surprise since the gcc family is built for optimal portability while the ICC compilers probably know the used CPU best since programmed by the CPU's manufacturer. However, test runs using Windows and the Visual Studio 6.0 compilers revealed another rather surprising speed-up of about 7% using the same hardware. Details may be found in [46].

Compression factors. Table 6.2 lists the accumulated compression factors achievable using the previously introduced compression scheme. These ratios have been determined using the same hardware setting specified before. A complete description of the test data along with a discussion of result images and charts will be given in Section 7.7 on page 154. However, the factors given correspond to data wavelet-transformed to the maximum possible level using the Daubechies-4 basis and preserving 80% of the signal's energy after coefficient thresholding.

As is clearly demonstrated by the examples given in Section 7.7, transformations to the maximum possible level are definitively worth the additional computational overhead regarding the compression ratio achievable when compared to level-1 transforms only. Moreover, the thus generated additional costs in terms of computation time are very limited since the overall costs are clearly dominated by the transform to the first level (Section 5.4.1).
6.2 COMPRESSION

TABLE 6.2

<table>
<thead>
<tr>
<th>Compression factors</th>
<th>In–core</th>
<th>File</th>
<th>File compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1:6</td>
<td>1:65</td>
<td>1:188</td>
</tr>
</tbody>
</table>

The results given in Section 7.7—and Figure 7.38, in particular—additionally show a weak dependence of achievable compression factors on the chosen wavelet basis which are the Haar and the Daubechies–4 basis as was already mentioned in Section 4.3.3. Of course, differences would be more apparent for wider wavelet bases which have not been chosen for reasons concerning computation time.

Again, note that the compression scheme is not based on requantization of thresholded wavelet coefficients. The main focus of the compression module, however, lies on in–core data and the algorithms interacting with compressed data without any restrictions.

Nevertheless, the compression ratios achievable are competitive. The compression based on vector quantization employed in the LFR publication [75] achieves a ratio of 1:24 that may be improved up to 1:120 with an additional Lempel–Ziv coder. Besides, attainable compression factors for wavelet–based compression schemes designed for light field data are ranging from an order of magnitude of 10 in [71] without requantization of thresholded coefficients, too, up to orders of magnitude of 100 in [98] and even 1000 in [83]. As a matter of course, the latter two do employ requantization methods. However, note that none of these approaches offers support for a dynamic setting.
All the visual examples and performance analyses given so far are confined to synthetic scenes only, i.e. all the input imagery has been previously generated by the use of raytracing software. An evaluation in a real-world setting, however, should make up an essential part of the analysis of the system's capabilities—particularly with regard to the possibility of a future integration of the system into an application that acquires image data directly from a hand-held video camera.

On the one hand, the evaluations compiled in this Chapter 7 focus on the processes during the core's second mode, the so-called oracle mode, using real image material read from a video camera as input material. The effects of local updates are analyzed in detail in Section 7.4. Moreover, several examples demonstrating the core's selective decision to accept or refuse new image information are given in Section 7.5.

Section 7.4, however, is further cut into two parts, presenting the effects of an incremental update operation at the update position itself at first (Section 7.4.1), followed by a detailed discussion of such an update's influences in its local vicinity (Section 7.4.2).

On the other hand, Section 7.6 discusses a very important analysis, namely the dependency of the overall quality of reconstructed views on the amount of available information, i.e. the number of images inserted into the representation. Finally, the impacts of only moderate and more aggressive compression on the visual quality are illustrated in Section 7.7.

Prior to the presentation of the evaluations themselves, a few prerequisites are introduced and discussed. At first, Section 7.1 gives an overview of the special-purpose equipment and its setup that was used for all the evaluations. Furthermore, the use of a real video camera requires to perform a calibration procedure in order to know about the optical system. Accordingly, Section 7.2 discusses any relevant calibration issue as well as the important parameters in the present setting. Section 7.3 additionally introduces a few error measures used later on in this Chapter 7.
7.1 SETUP

All real-world evaluations of the system have been carried out with imagery recorded using a motorized object rig with a video camera attached to it. Such a computer-controlled acquisition system proved to represent a passable compromise to a hand-held camera for evaluation purposes, since, given the object rig, tracking of the camera’s pose does not impose a hard problem anymore. How to determine the current position and orientation of the camera, i.e. its pose, is stated in Section 7.2.

The construction used is a Kaidan Magellan 2500, an object rig that is conceptually able to cover a full sphere around an object at its center. Figure 7.1 a) shows a promotional image of the design. The object of interest is placed on the turntable. The camera is mounted on the robot’s arm that needs to be equilibrated as good as possible. Note that the arm’s freedom of movement is restricted to an upright plane, i.e. its rotation axis is horizontally arranged, as is indicated by the horizontal arrows in Figure 7.1.

Moving the arm of the Magellan 2500 in its plane and rotating the object of interest on the turntable—the turntable’s rotation axis is indicated by the vertical arrows in Figure 7.1—offers the possibility to view the object from all possible angles, in principle. Of course, positions lower than the turntable itself are not meaningful because the object gets partially or completely hidden by the turntable. Moreover, the robot’s arm may collide with parts of its own construction and/or with the turntable’s mount. Note that the person or the software operating the Magellan 2500 is responsible to detect and avoid such situations since the robot itself does know precisely nothing about its state and position.

Figures 7.1 b) and c) show the object rig and some additional lighting equipment actually used for the evaluations. Note that the blue cloths are utilized for background subtraction operations needed in the connexion of other projects. However, they provide an accurately defined background in the present context but are not used for any other purposes at all.

The camera being used for image recording is a Sony DFW-X700 camera, as is shown in Figure 7.2. The camera features a singular CCD sensor chip with XGA resolution and an IEEE 1394 FireWire interface for communication and data transfer.
FIGURE 7.2 Sony DFW-X700 camera with XGA 1024x768 CCD resolution and FireWire interface.

The image acquisition process is controlled using a custom software package. It is a reimplemention of the software provided by Kaidan, but targeted at Linux systems. It offers the same functionality regarding positioning and controlling as the bundled software does, but also integrates everything necessary to handle the video stream by way of a FireWire interface.

Note that, although designed for, the video information is not directly fed into the system's core running in the oracle mode, mainly due to platform constraints. Configured for the video mode (Section 3.1) the system reads video information from a video interface available on an SG1 computer system only. With an additional FireWire interface adapter for IRIX platforms, however, directly reading from an on-line camera could easily be done, too. As a consequence of these constraints, the image processing pipeline is set up as follows. Images are being read from the video camera, dependent on the robot's current position, and stored on disk. The system's core, configured for batch mode acquisition (Section 3.1), thereafter reads from this stack of available real-world images.

7.2 CALIBRATION

Calibration of a camera and its optical system is unavoidable in a real-world setting. This applies to video cameras as much as to still cameras. An intrinsic calibration needs to be executed in any case in order to know about the optical parameters of the system in use, such as the focal length, for instance. An extrinsic calibration may additionally be necessary, if no other way of camera pose estimation is available that is accurately enough.

Note that an intrinsic calibration may be done with a small set of images of an appropriate calibration object. The parameters learned from the intrinsic calibration procedure describe the optical system and do therefore apply to all subsequently recorded images.

An extrinsic calibration, in contrast, needs to be performed on every single shot, since the set of parameters thus determined do not describe the optical system but the camera's pose corresponding to the respective picture.

Intrinsic calibration. The intrinsic parameters of the previously mentioned camera and its lenses are determined using the Caltech Calibration Toolbox [10] which is a software package to be used with MATLAB. This calibration procedure is based on the methods derived in [58] and [146]. It uses a planar chessboard pattern as calibration object. Figure 7.3 shows an example of an image series that needs to be provided as input to the
intrinsic calibration procedure. The white circular area inside one of the black squares helps to identify the plane’s orientation.

A complete documentation of the Caltech Calibration Toolbox as well as a tutorial on camera calibration with directions for further reading may be found in [10]. An implementation of this toolbox in the C language is also included in the Open Source Computer Vision library [62] freely distributed by Intel.

**Extrinsic calibration.** Extrinsically calibrating a large amount of images is preferably avoided for the following two reasons. Firstly, the computational overhead is significant, especially for large series, and may not be reasonably operable if not fully automated. Secondly and even more importantly, the calibration object needs to be visible in every single image such that the calibration procedure may detect it precisely enough.

Fortunately, the Magellan robot described in the previous Section 7.1 may be exactly positioned by specifying two parameters, namely \( \phi \) defining the rotation angle of the turntable, and \( \theta \) defining the rotation angle of the robot’s arm. Both parameters must be specified with respect to a self-defined point of origin. However, the robot proved to stop very precisely at a desired position, as is illustrated in Figure 7.4 below. It compares the parameter \( \theta \) provided by the control software with the results from extrinsic calibrations of images of two test series. Note that the above mentioned Caltech Calibration Toolbox also computes extrinsic calibration parameters and has therefore been adopted for these two test series as well.

**FIGURE 7.4** Comparison of positional parameters: Theta angle \( \theta \) provided by the extrinsic image calibration vs. Magellan 2500 object rig.

a) First experiment.

b) Second experiment.
Preceding the tests the object rig was adjusted such that the center of the upper hemisphere the camera can potentially cover resides in the middle of the turntable, i.e. the upright position of the robot's arm conforms to $\vartheta = 90.0^\circ$ and correspondingly, the plane satisfying $\varphi = 0.0^\circ$ coincides with the turntable.

The parameter of interest in both experiments is the angle $\vartheta$ because the arm with the camera attached to it shows—although well equilibrated—remarkable oscillating movements after a stop, especially for $\vartheta < 45.0^\circ$. Hence, if the $\vartheta$ parameter provided by the control software is accurate enough, in spite of the swinging, then the $\varphi$ parameter will be as well, for the turntable's movements are slow and smooth.

Both plot series of both experiments shown in Figure 7.4, however, exhibit a linear behavior with almost no deviation from each other, i.e. the construction with the camera attached to it moves to and stops at a desired position very precisely, as can be verified with extrinsic calibration procedures. Moreover, as is illustrated in Figures 7.5 and 7.6 below, the absolute difference stays within $\pm 0.5^\circ$ for $\vartheta \geq 30.0^\circ$ for the values of the first test series and for $\vartheta > 40.0^\circ$ for the second test. The drastic increase of the relative error for smaller values of $\vartheta$, however, is an awaited consequence of the reference value getting continuously smaller.

**FIGURE 7.5** Difference measurements of the first experiment in Figure 7.4.
a) Absolute difference values.
b) Relative difference values.

**FIGURE 7.6** Difference measurements of the second experiment in Figure 7.4.
a) Absolute difference values.
b) Relative difference values.
Figure 7.7 finally gives evidence that the extrinsic calibration method may be used as the reference for camera pose estimation. It plots the accuracy of the calibration procedure in terms of pixel errors in both dimensions of the image plane. The errors clearly stay within the bounds of one pixel, in case of the second experiment even within half a pixel.

**FIGURE 7.7** Comparison of the calibration pixel errors of the two experiments shown in Figure 7.4.  
a) First experiment (Figure 7.4, a).  
b) Second experiment (Figure 7.4, b).

Since LF-like representations are not extremely sensitive regarding preciseness of the camera pose of their input images—quite contrary to image-based methods reconstructing geometry, such as 3D video coding [140], for instance—the accuracy attainable using the Magellan robot's positioning abilities is sufficient enough in the present context, facing the given test series. In other words, it is not inevitable to perform an extrinsic calibration on every single acquired image.

The following Figures 7.8–7.11 show illustrations and reconstructed views of four different light fields, constructed from image sets acquired with the equipment introduced in Section 7.1. Images a) show reconstructions of crudely sampled light fields whereas images b) show a more densely sampled version. Note that all these reconstructions have been computed form the eMRA representation, after completing the core's approximation mode, using a local reconstruction operator.

**FIGURE 7.8** The Frog light field.  
a) A crudely sampled and interpolated light field, built using 25 input images.  
b) A more densely sampled and interpolated light field, using 90 input images.  
c) 3D visualization of the input camera poses of the 25 input views used for image a).  
d) 3D visualization of the input camera poses of the 90 input views used for image b).
FIGURE 7.9 The Husky light field.
  a) A crudely sampled and interpolated light field, built using 90 input images.
  b) A more densely sampled and interpolated light field, using 340 input images
  c) 3D visualization of the input camera poses of the 90 input views used for image a).
  d) 3D visualization of the input camera poses of the 340 input views used for image b).

FIGURE 7.10 The Ganesha light field.
  a) A crudely sampled and interpolated light field, built using 64 input images.
  b) A more densely sampled and interpolated light field, using 256 input images
  c) 3D visualization of the input camera poses of the 64 input views used for image a).
  d) 3D visualization of the input camera poses of the 256 input views used for image b).

FIGURE 7.11 The Haathi light field.
  a) A crudely sampled and interpolated light field, built using 64 input images.
  b) A more densely sampled and interpolated light field, using 256 input images
  c) 3D visualization of the input camera poses of the 64 input views used for image a).
  d) 3D visualization of the input camera poses of the 256 input views used for image b).
Images c) and d) of Figures 7.8–7.11 give coarse 3D illustrations of the camera’s poses for all sets of input images. Each red dot represents a different COP whereas the connected green line segment corresponds to the respective camera’s direction of view. The yellow sphere is a coarse placeholder for the object of interest, residing at the origin of the white coordinate system. All images a) and b) of Figures 7.8 through to 7.11 show reconstructions as seen from the initial position in the middle of the xy parameter space which do not coincide with a grid location. According to the exact parameter sets of the four example light fields listed in Tables 7.1 and 7.2 below, these initial positions correspond to \( x = 3.5, y = 3.5 \) for the Frog light field (Figure 7.8) and to \( x = 7.5, y = 7.5 \) for the remaining three light fields shown in Figures 7.9–7.11.

The sets of input images the given examples of light fields are constructed from are described in Appendix B with Tables B.3–B.6 completely listing all relevant parameters. Note that the examples of light fields shown in the remainder of this Chapter 7 are all based on one of these four sets of input imagery described in detail in Tables B.3–B.6.

**TABLE 7.1** Parameter set describing the Frog (Figure 7.8) and the Husky light field (Figure 7.9).

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Frog light field (Figure 7.8)</th>
<th>Husky light field (Figure 7.9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameterization model</td>
<td>Sphere-plane</td>
<td>Sphere-plane</td>
</tr>
<tr>
<td>Angular spacing (( \Delta \theta, \Delta \phi ))</td>
<td>((3.5^\circ, 4.29^\circ))</td>
<td>((6.6^\circ, 2.0^\circ))</td>
</tr>
<tr>
<td>Number of input images used</td>
<td>25 (Image a), 90 (Image b)</td>
<td>90 (Image a), 340 (Image b)</td>
</tr>
<tr>
<td>Interpolation filter</td>
<td>Gauss, 5-tap</td>
<td>Gauss, 5-tap</td>
</tr>
<tr>
<td>Wavelet basis</td>
<td>Haar</td>
<td>Haar</td>
</tr>
<tr>
<td>Maximum transformation level</td>
<td>( M = 2 )</td>
<td>( M = 3 )</td>
</tr>
</tbody>
</table>

**TABLE 7.2** Parameter set describing the Ganesha light field (Figure 7.10) and the Haathi light field (Figure 7.11).

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Ganesha light field (Figure 7.10)</th>
<th>Haathi light field (Figure 7.11)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light field parameters</td>
<td>( 10 \times 16 \times 256 \times 256 )</td>
<td>( 10 \times 16 \times 256 \times 256 )</td>
</tr>
<tr>
<td>Parameterization model</td>
<td>Sphere-plane</td>
<td>Sphere-plane</td>
</tr>
<tr>
<td>Angular spacing (( \Delta \theta, \Delta \phi ))</td>
<td>((2.0^\circ, 2.0^\circ))</td>
<td>((2.0^\circ, 2.0^\circ))</td>
</tr>
<tr>
<td>Number of input images used</td>
<td>64 (Image a), 256 (Image b)</td>
<td>64 (Image a), 256 (Image b)</td>
</tr>
<tr>
<td>Interpolation filter</td>
<td>Gauss, 5-tap</td>
<td>Gauss, 5-tap</td>
</tr>
<tr>
<td>Wavelet basis</td>
<td>Haar</td>
<td>Haar</td>
</tr>
<tr>
<td>Maximum transformation level</td>
<td>( M = 3 )</td>
<td>( M = 3 )</td>
</tr>
</tbody>
</table>
7.3 ERROR MEASURES

Besides the equipment and its setup (Section 7.1) and the issues about camera calibration (Section 7.2) there is a third component the evaluations given in this Chapter 7 rely on. In order to (1) judge on the effectiveness of local updates, (2) demonstrate the selectiveness during the oracle mode, (3) evaluate the overall reconstruction quality dependent on the amount of available information and (4) quantify the impact of the compression mechanism on the visual reconstruction quality, error and other measures will be heavily used. All of them are listed and briefly commented on below.

Regardless of the fact that all of these measures are well-known and, hence, widely used, the exact method of calculating the actual values will be given as well. Awkwardly, many different ways of computing some of these measures do exist, which mainly applies to the class of signal-to-noise ratios. Moreover, it often remains unclear how the measures are being calculated in case of multi-spectral data such as color images, for instance. Thus, detailed formulae of the measures used in the context of this thesis supplement the following list which is divided into (1) difference, (2) energy and (3) signal-to-noise measures.

**Difference measure.** A difference measure is used solely for images of three channels, i.e. RGB images. Differences are computed on each channel separately as is shown in Expression 7.1 below, with $p_i$ being the $i$th pixel value and $N$ denoting the total number of pixels available in an image. The three channels of difference values are again combined to form the result referred to as the difference image.

$$\left| p_{i,2} - p_{i,1}\right|_{R,G,B} \quad i = 1, \ldots, N \quad (7.1)$$

**Energy measures.** The energy of some signal $f(x)$ is computed as is shown in Equation 7.2 below with its continuous version on the left and its discrete version on the right hand side (see also Equation 4.1 on page 53).

$$E_{\text{continuous}} = \int_{-\infty}^{+\infty} |f(x)|^2 \, dx \quad E_{\text{discrete}} = \sum_j |f(x_j)|^2 \quad (7.2)$$

Accordingly, the energy of some image is a discrete sum of pixel values. The energy of multi-channel images, in turn, is computed as the sum over all channels as is stated in Equation 7.3 below. Note that an energy value based on a luminance signal only may be meaningful, too, which is in fact widely used.

$$E_i = \sum_{R,G,B} \left( \sum_{i = 1}^{N} |p_i|^2 \right) \quad (7.3)$$

Following the above formula, the energy of a single pixel of some multi-channel image may be determined as is shown in Equation 7.4 below.

$$E_p = \sum_{R,G,B} |p_i|^2 \quad i \in [1, N] \quad (7.4)$$
Apart from energy values of images and sole pixels, the energy of a complete light field data set is needed, too. Such an energy in case of wavelet-transformed light field data may be calculated as is stated in Equation 7.5 (see also Equation 4.39 on page 65). Again, multiple channels are addressed in an additive manner.

\[
E_i = \sum_{R,G,B} \left( \sum_k |c_{Mk}|^2 \right) + \sum_{R,G,B} \left( \sum_{m=1}^M \sum_k |d_{mk}|^2 \right)
\]  

\[\text{(7.5)}\]

**Signal-to-noise ratios.** The comparison of the energy a signal carries with the amount of noise it contains therein is a common and quite popular way of quantifying the quality of a signal of any kind. Such a relation is measured by properly comparing a reconstructed, degraded or modified signal with its original version, resulting in a single value that reflects the quality of the modified signal.

The metric that is heavily used in the field of processing and coding of video and image data is the peak signal-to-noise ratio (PSNR). For a single-channel image with a maximum pixel value of 255 the PSNR may be computed as is shown in Equation 7.6 below. Note that a variety of slightly different calculation methods may be found as well.

\[
PSNR_{\text{dB}} = 10 \cdot \log_{10} \left( \frac{255^2}{MSE} \right) = 20 \cdot \log_{10} \left( \frac{255}{RMSE} \right)
\]

\[\text{(7.6)}\]

\[
\text{with} \quad MSE = \frac{\sum_i (p_{i,2} - p_{i,1})^2}{N} \quad \text{RMSE} = \sqrt{MSE}
\]

\[\text{(7.7)}\]

The PSNR value is mostly determined based on a luminance signal only. In case of multi-channel images with no explicit luminance component, such as the RGB format, the luminance may be computed as is shown in Expression 7.8.

\[
(0.3 \cdot p_{i,R}) + (0.59 \cdot p_{i,G}) + (0.11 \cdot p_{i,B}) \quad i \in [1, N]
\]

\[\text{(7.8)}\]

Despite their broad application, signal-to-noise ratios do not necessarily equate with the human subjective perception, i.e. a higher SNR/PSNR value does not necessarily comply with better visual quality. As a consequence, absolute values are not always meaningful but the comparison among multiple, equally computed values is.

The signal-to-noise measure applied in the context of this thesis is used solely for internal comparison purposes and is therefore kept simple to compute. It is a pure SNR value that takes all color channels into account, all of them evenly weighted, as is shown in Equation 7.9, with \(E_{\text{signal}}\) being the \(E_i\) (Equation 7.3) of the signal to be analyzed and \(E_{\text{noise}}\) the \(E_i\) of the noise, i.e. the difference (Equation 7.1) between a reference signal and the signal to be examined.

\[
\text{SNR}_{\text{dB}} = 10 \cdot \log_{10} \left( \frac{E_{\text{signal}}}{E_{\text{noise}}} \right)
\]

\[\text{(7.9)}\]

Further readings concerning signal-to-noise measures and image quality computation in general may easily be found, e.g. [97, 103].
7.4 EFFECT OF LOCAL UPDATES

Since the camera attached to the robot's arm moves on a spherical surface it is the spherical parameterization (Section 6.1.2) that is most preferably used for the input imagery acquired with the Magellan object rig. Consequently, all example light fields shown in the following sections are parameterized applying the spherical setting. Note that the two angular parameters $\varphi$ and $\theta$ provided by the robot's control software may directly be used as the two polar angles of the 3-dimensional polar coordinate system. The third parameter, i.e. the radius, can be computed through extrinsic calibration of one or more images right after having settled the robot.

Moreover, if not stated otherwise, the approximation mode for the construction of all examples was completed using a 5-tap Gaussian interpolation filter and the Haar wavelet basis. The transformation to difference space was done to the maximum possible depth.

Before moving on to the results captured during the oracle mode, Figure 7.12 again demonstrates the capabilities of the hierarchical scattered data interpolation (HSDI) performed during the construction of the eMRA. Image b) shows the result after the completion of the approximation mode for the same viewpoint as for image a). Note that this viewpoint obviously lay inside a NO_DATA area. Image d), in contrast, resided at the border of such an area since some input image data is within the support of the basis functions centered at this viewpoint. Image c) gives the result of the HSDI process. Again, note the ghosting artifacts of neighboring views as was discussed in Section 4.9.

FIGURE 7.12 Effect of the HSDI using a 5-tap Gaussian as interpolation filter, shown for 2 examples.

a) Rendering before the run of the HSDI, showing no data (1st example).
b) Result of the HSDI, displaying the same viewpoint as a).
c) Result of the HSDI, displaying the same viewpoint as d).
d) Rendering before the run of the HSDI, showing very little data (2nd example).
7.4.1 Influence at Update Position

The effects of a local update operation at the update position itself have already been demonstrated in Section 4.9, using raytraced input imagery. The following example is based on real images.

Figure 7.13 illustrates a local update of a light field that was initially constructed from only 60 randomly selected input images of the Haathi scene (Table B.6 on page 172). The complete set of parameters of this example light field is given in Table 7.3, which also applies to the preceding Figure 7.12.

Image a) of Figure 7.13, however, shows a reconstruction of the light field for the same viewpoint as for image b), some new input image. It is important to note that this new input image has not been used for the initial construction of the light field. Image c) shows the difference information of b) with respect to image a). Image d) finally shows a reconstruction for the same viewpoint as was used for image a) after the local update operation with c) as input has been performed.

FIGURE 7.13 Effects of a single local update operation.
a) Interpolated reconstruction, after the HSDI, as seen from the same viewpoint as b).
b) New input image.
c) Difference image of b) to a).
d) Reconstructed view after the local update operation that used c) as input.
e) False color coding (linear mapping, scaled domain) of the difference image c).
f) False color coding (logarithmic mapping, scaled domain) of the difference image c).
g) Color spectrum used by the false color coder.
Due to the linear basis functions employed such perfect reconstructions as is shown in Figure 7.13 d) are feasible at grid locations of the \( xy \) parameter space only. Clearly, at least one input image with its COP coinciding with a sampling point of the parameterization grid must be available in order to perform the local update operation. However, in case of the example light field shown in Figure 7.13 this applies to all input images, i.e. the COPs of all available input images match with a grid location of the \( xy \) parameter space which has been forced by the light field’s resolution in \( x \) and \( y \) (Table 7.3) meeting the acquisition grid of the input image set (Table B.6).

Images c) and f) of Figure 7.13 additionally give false-color coded versions of the difference image c). These false-color images encode the differences using the color spectrum between red and green. To be more precise, the \( E_p \) values (Equation 7.4) of the pixels in the difference image are recolored such that the maximum pixel energy value corresponds to red and the minimum value to green. Intuitively speaking, red means strong differences, green little or no differences at all. The color spectrum employed is additionally given in Figure 7.13 g).

If this mapping is done linearly, the resulting false-color image will mostly be greenish. A domain scaling improves the visualization. That is, mapping the target color range to the interval from the minimum to the maximum occurring \( E_p \) value—instead of the maximum possible value—better visualizes the differences. Moreover, an even more clear result may be achieved by using a logarithmic mapping instead of a linear mapping. The linear mapping is shown in image c) of Figure 7.13 whereas image f) shows the logarithmic mapping. Note that both visualizations combine the mapping with a scaled domain.

### Table 7.3
Parameter set describing the example light field used in Figures 7.12 and 7.13.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Example light field of Figures 7.12 and 7.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light field’s resolution</td>
<td>( 32 \times 32 \times 256 \times 256 )</td>
</tr>
<tr>
<td>Parameterization model</td>
<td>Sphere–plane</td>
</tr>
<tr>
<td>Angular spacing (( \Delta \theta, \Delta \phi ))</td>
<td>( (1.0^\circ, 1.0^\circ) )</td>
</tr>
<tr>
<td>Set of input images used</td>
<td>Haathi (Table B.6)</td>
</tr>
<tr>
<td>Number of input images used</td>
<td>60</td>
</tr>
<tr>
<td>Interpolation filter</td>
<td>Gauss, 5-tap</td>
</tr>
<tr>
<td>Wavelet basis</td>
<td>Haar</td>
</tr>
<tr>
<td>Maximum transformation level</td>
<td>( M = 4 )</td>
</tr>
</tbody>
</table>

#### 7.4.2 Influence in Local Vicinity

Clearly, any update performed is expected to not only improve the 4-dimensional representation at the update position itself but rather in a local neighborhood, according to the basis functions employed.
There are, however, quite a few different alternatives of how to verify any enhancements of a single local update operation in a local vicinity, by way of measurement as well as through visual clues. The following list briefly introduces the options chosen and discussed subsequently.

1) **Signal energy:**
   Computing signal energies of reconstructed views according to Equation 7.3 before and right after a local update and building the differences between these two sets will give evidence of where changes occur.

2) **Signal-to-noise ratio (SNR):**
   Changes may also be detected by computing pre- and post-update SNR values. As a matter of course, additionally acquired reference images are necessary to calculate the SNR measures at the respective positions.

3) **Visual verification:**
   Visual evidence may be obtained by inspecting the result images themselves, difference images or false-color coded difference images, too.

All of the above possibilities have been applied to an example light field based on the Ganesha scene (Table B.5 on page 171). The complete set of parameters this light field was built with is given in Table 7.4 below. The local update operation, however, was performed in the middle of the xy parameter space at the position $x = 7.5$, $y = 7.5$. Note that this update position does not coincide with a grid location of the xy parameter space.

Using this setting, two different experiments have been conducted. Figure 7.14 a) visualizes the situation for the first experiment, with the red dot representing the update position. The light field used for this first experiment was initially constructed from all 256 available input images, i.e. one input image per grid point, represented by the green dots in Figure 7.14. Since this light field is fairly densely sampled, the visual clues of the update are not expected to be rich.

---

**TABLE 7.4** Parameter set describing the example light field used in Section 7.4.2.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Example light field of Section 7.4.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light field’s resolution</td>
<td>$16 \times 16 \times 128 \times 128$</td>
</tr>
<tr>
<td>Parameterization model</td>
<td>Sphere-plane</td>
</tr>
<tr>
<td>Angular spacing $(\Delta \phi, \Delta \theta)$</td>
<td>$(2.0^\circ, 2.0^\circ)$</td>
</tr>
<tr>
<td>Set of input images used</td>
<td>Ganesha (Table B.5)</td>
</tr>
<tr>
<td>Number of input images used</td>
<td>256 (1st experiment), 240 (2nd experiment)</td>
</tr>
<tr>
<td>Interpolation filter</td>
<td>Gauss, 5-tap</td>
</tr>
<tr>
<td>Wavelet basis</td>
<td>Haar</td>
</tr>
<tr>
<td>Maximum transformation level</td>
<td>$M = 3$</td>
</tr>
</tbody>
</table>
Figure 7.14 b) illustrates the test positions for the SNR measurements. An additional set of nine reference image has been acquired for all of these positions, which are marked by the red dots. It is again important to note that these nine reference images have not been used for the construction of the light field. Furthermore, in order to account for potential sources of error due to the non–perfect positioning of the robot's arm, ten such sets of nine reference images each have been acquired.

Figure 7.14 c) finally visualizes the situation for the second experiment. The light field used in this case was constructed from 240 input images. The ones that were omitted in comparison with the first experiment are the direct neighbors of the update position.

Figure 7.15 shows visualizations of the differences of signal energies, i.e. $E_i$, for the first experiment. Before the local update was performed $15^2 \times 225$ images were reconstructed and stored together with their individual energy $E_i$ (Equation 7.3). These views were reconstructed in the intervals $x, y \in [4, 11]$ with increments of 0.5, which is visualized by the thin grid lines in images c) and d) of Figure 7.15. After the local update operation at $x = 7.5, y = 7.5$ the same 225 images were again reconstructed and stored with the anew computed energies. Building the differences of these two sets of signal energies leads to visualizations as they are shown in Figure 7.15. Firstly note that changes do occur in a local neighborhood of the update position. Secondly, the respective vicinity is bounded to the area potentially influenced by the quaquad–linear basis functions.

Besides energy measures, a signal–to–noise ratio may also be used to visualize any improvements in a local neighborhood of an update operation. Equation 7.9 gives the method of computation actually used. Note that the absolute values of the SNR measure employed are not implicitly meaningful as they are used exclusively for internal comparison (Section 7.3). In fact, the resulting absolute values are much higher as compared to the PSNR measure given in Equation 7.6, for instance.

Again, the noisy signal used for an SNR measure may be extracted by computing the difference between a reference and the signal being analyzed, i.e. between a reference image acquired by the video camera (Figure 7.14 b) and a reconstructed view.
FIGURE 7.15  Visualizations of differences of signal energies of extracted images, located around the update position in the setting of the first experiment (Figure 7.14 a).

a) Local vicinity around the update position at \( x = 7.5, y = 7.5 \).
b) Top view of a).
c) Bigger vicinity of a).
d) Top view of c).

Figure 7.16 visualizes the measured SNR values computed for all nine reference images, averaged over all ten sets. The beige surfaces thereby represent pre-update measures whereas the reddish surfaces depict the post-update values. A strong gain in terms of signal-to-noise ratio is clearly visible, especially at the update position in the center. Nonetheless, it is important to also note that a slight degradation may happen in the local neighborhood of the update position, as is visible at the right side of Figure 7.16 b), for instance. The reason for this effect are the very limited interpolation abilities of the quadra-linear basis functions applied, specifically in such a densely sampled light field.

The exact numbers of the SNR computations following Equation 7.9 are compiled in Table 7.5. According to these values, the single local update operation causes an increase of about 5.4 dB at the update position, averaged over all ten reference sets.

Figure 7.17 additionally gives a qualitative comparison of all SNR values of the ten different sets of reference images which are all very similar. Visualization differences of size
and view points are due to the non automated processing of measured data. The actual values, however, do only differ marginally. Figure 7.18 shows the SNR gain compared over all ten sets, again measured at the update position. Note that the difference between pre- and post-update SNR is about the same for all reference sets, although the absolute values slightly vary. Thus, any potentially imprecise camera positioning caused by the acquisition setup (Sections 7.1 and 7.2) does not heavily influence the result, i.e. it is not dependent on possibly imprecise camera pose data of input images.

A visual verification at the update position itself as well as in its local vicinity may additionally be done. However, post-update versions of reconstructed views proved to differ little from their corresponding pre-update version, again due to the pretty densely sampled test light field.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>pre-update</th>
<th>post-update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average SNR</td>
<td>55.2917</td>
<td>60.6927</td>
</tr>
<tr>
<td>Minimal SNR value</td>
<td>54.382</td>
<td>59.407</td>
</tr>
<tr>
<td>Maximal SNR value</td>
<td>56.156</td>
<td>61.82</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.6876</td>
<td>0.9663</td>
</tr>
<tr>
<td>SNR gain (average dB)</td>
<td>5.401</td>
<td></td>
</tr>
</tbody>
</table>
FIGURE 7.17  Visualizations of SNR measurements [dB] of all 10 reference sets.  
Left 2 columns: Top view of reference sets 1 to 10.  
Right 2 columns: Side view of reference sets 1 to 10.
The same analyses as they have been presented for the setting of the first experiment (Figure 7.14 a) have also been conducted with the second setting (Figure 7.14 c). However, the improvements due to the local update operation are expected to be more significant in comparison to the first experiment since the update happens in coarsely sampled area.

Figure 7.19 visualizes the differences of signal energies according to Equation 7.3, computed for the second experiment. The generation of these difference values was done in the very same way as in the first setting. If comparing the results shown in Figure 7.15 and those presented in Figure 7.19, they prove to be very similar from a qualitative point of view. However, the peaks differ one order of magnitude, i.e. a local update operation generates more impact in a less densely sampled region, as was expected.

Regarding SNR measure, the above mentioned fact gets confirmed, too. Figure 7.20 shows the SNR values computed for one reference set only. Again, the beige surface represents the pre–update measures whereas the reddish surface depicts post–update values. Note that, in contrast to Figures 7.16 and 7.17, the whole neighborhood of the local update position gets enhanced, i.e. no degradation occurs any more. In simpler words, the two surfaces in Figure 7.20 do not touch or even intersect each other but are all well separated.

The exact SNR values computed according to Equation 7.9 are listed in Table 7.6. Again, the values shown correspond to the update position itself. Because the results of all ten reference sets used in the setting of the first experiment are very alike (Figure 7.17) only one set was utilized during the second experiment. As a consequence, Table 7.6 lacks the statistics of Table 7.5. However, the increase at the update position is slightly higher as was the case for the first experiment, namely about 6 dB, which is due to the generally smaller SNR values because of the more crudely sampled surrounding of the update position in the setting of the second experiment.
FIGURE 7.19 Visualizations of differences of signal energies of extracted images, located around the update position in the setting of the second experiment (Figure 7.14 c).
   a) Local vicinity around the update position at $x = 7.5, y = 7.5$.
   b) Top view of a).
   c) Bigger vicinity of a).
   d) Top view of c).

FIGURE 7.20 Two visualizations of the same SNR measurements [dB] of extracted images located around the update position (2nd experiment, Figure 7.14 c).
Contrary to the first experiment, the improvements are also clearly visible in reconstructed views, namely at the update position itself but also in its local vicinity, as is shown in Figures 7.21 and 7.22. As a matter of course, the changes are also apparent in corresponding difference images not shown at this place.

Figure 7.21 shows reconstructed views before the local update operation was performed. The central image corresponds to the update position whereas the outer four views were reconstructed at the four grid locations in the \( xy \)–parameter space surrounding the update position. Once again, note the severe ghosting artifacts which are blended in by the HSDI during the construction of the eMRA (Section 4.9).

Figure 7.22 shows reconstructions for the very same five positions right after the update operation has been performed, revealing the Ganesha statue in all views. Note that perfect reconstructions, e.g. Figure 7.13 on page 136, are possible for updates at an exact sample location in the \( xy \)–parameter only due to the quadra-linear basis functions employed as was already mentioned in Section 7.4.1.

### Table 7.6

<table>
<thead>
<tr>
<th>Criterion</th>
<th>pre-update</th>
<th>post-update</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNR value</td>
<td>26.627</td>
<td>32.656</td>
</tr>
<tr>
<td>SNR gain (dB)</td>
<td>5.984</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5 Selectivity of Incremental Updates

During the oracle mode, the system's core continuously grabs new input images, aiming at the progressive refinement of the estimate built throughout the approximation mode. However, the core does not simply insert every new information presented. It rather decides on the acceptance of each input image individually, based on the difference between the input image and the corresponding reconstructed view (Section 3.3.2). Once an image has been accepted based on the mechanism discussed in Section 4.8 the new information is incrementally inserted using the local update procedure (Section 4.7).

The acceptance criterion used for the following examples applies a simple threshold to the RMSE value computed on the difference between the presented input image and the corresponding reconstructed view. Clearly, the threshold parameter needs to be specified from case to case, based on the given setting and the light field parameters.

The selectivity of incremental updates is illustrated using a test light field based on the Husky scene (Table B.4 on page 171) with its resolution set to \( 16 \times 16 \times 256 \times 256 \). It was initially constructed from 300 input images randomly selected from the pool of available imagery. The complete set of parameters for this test light field is compiled in Table 7.7.
FIGURE 7.21 Pre-update images at the four grid points surrounding the local update position plus at the update position itself (2nd experiment, Figure 7.14c).

FIGURE 7.22 Post-update images, corresponding to the ones in Figure 7.21.
After the completion of the approximation mode, the core again randomly selects previously unused images and decides on their acceptance during the oracle mode. It is important to note that the grid of input images does not match with the \( xy \) parameterization grid (compare Table 7.7 and Table B.4), i.e. no COP of an input image coincides with a sample location in the \( xy \) parameter space, neither of images from the initial input set nor of images used to demonstrate the selective acceptance.

Figures 7.23–7.25, however, give examples of accepted input images. Image a) in each case shows the new input image, b) the corresponding reconstructed view and c) the corresponding reconstruction after the local update operation. All of these incremental updates clearly improve on the quality of the local reconstruction.

Figures 7.26–7.28, in contrast, show examples of rejected input images, i.e. input images the system’s core decided not to insert, based on the RMSE value computed. As far as the visual quality is concerned, the reconstructions shown in images b) are in fact satisfying. However, note that the reconstructions shown in images c) reproduce the views already shown in images b) since no local update operations needed to be performed.

The RMSE values the core’s decision is based on are computed following the rules given in Equations 7.7 with the luminance signal computed as is shown in Expression 7.8 (Section 7.3). Furthermore, the update threshold \( \tau_f \) was set to \( \tau_f = 13.0 \) for all examples shown (Figures 7.23–7.28). Table 7.8 on page 150 gives the exact RMSE values computed for each of the examples presented, confronted with the update threshold \( \tau_f \).

Moreover, Table 7.8 also lists the corresponding RMSE values not based on the luminance but on all three color channels instead. That is, the MSE is computed according to Equation 7.7 for each channel independently and then summed up before building the final RMSE value. However, Table 7.8 shows that this alternative update threshold \( \tau_a \) set to \( \tau_a = 23.0 \) for the examples given might also be successfully used as decision criterion.

### Table 7.7

Parameter set describing the example light field used in Section 7.5.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Example light field of Section 7.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light field's resolution</td>
<td>( 16 \times 16 \times 256 \times 256 )</td>
</tr>
<tr>
<td>Parameterization model</td>
<td>Sphere–plane</td>
</tr>
<tr>
<td>Angular spacing (( \Delta \phi, \Delta \theta ))</td>
<td>( (6.0^\circ, 2.0^\circ) )</td>
</tr>
<tr>
<td>Set of input images used</td>
<td>Husky (Table B.4)</td>
</tr>
<tr>
<td>Number of input images used</td>
<td>300</td>
</tr>
<tr>
<td>Interpolation filter</td>
<td>Gauss, 5-tap</td>
</tr>
<tr>
<td>Wavelet basis</td>
<td>Haar</td>
</tr>
<tr>
<td>Maximum transformation level</td>
<td>( M = 3 )</td>
</tr>
</tbody>
</table>
FIGURE 7.23 Example (1) of an accepted local update operation (see Table 7.8 for RMSE values).

a) Input image.
b) Reconstructed view.
c) Updated view.

FIGURE 7.24 Example (2) of an accepted local update operation (see Table 7.8 for RMSE values).

a) Input image.
b) Reconstructed view.
c) Updated view.

FIGURE 7.25 Example (3) of an accepted local update operation (see Table 7.8 for RMSE values).

a) Input image.
b) Reconstructed view.
c) Updated view.
7.5 SELECTIVITY OF INCREMENTAL UPDATES

FIGURE 7.26 Example (4) of a rejected local update operation (see Table 7.8 for RMSE values).

a) Input image.
b) Reconstructed view.
c) Unchanged view.

FIGURE 7.27 Example (5) of a rejected local update operation (see Table 7.8 for RMSE values).

a) Input image.
b) Reconstructed view.
c) Unchanged view.

FIGURE 7.28 Example (6) of a rejected local update operation (see Table 7.8 for RMSE values).

a) Input image.
b) Reconstructed view.
c) Unchanged view.
7.6 DEPENDENCY ON FILLING DEGREE

A single local update operation causes a certain impact at the update position itself, but also in a local vicinity around the update position, as was discussed in the previous Section 7.4. Clearly, one local update will not bring forth a global impact since they are designed to be local.

However, having the basic idea of an easy-to-handle acquisition application in mind, it is important to analyze the global trends regarding overall reconstruction quality dependent on the amount of information included in the representation, i.e., its filling degree. It needs to be examined in what way the quality of any reconstructed view evolves—from a global point of view—dependent on the quantity of input images inserted.

The following experiment is again based on the set of 341 input images of the Husky scene (Table B.4). The resolution of all light fields constructed from this imagery are set to $32 \times 32 \times 128 \times 128$—the complete set of parameters is listed in Table 7.9. Randomly selecting a certain percentage of input images from the available set and constructing a light field with them allows to study the overall quality of reconstructed views from that light field, using the input set as reference images to compute SNR values. A meaningful statistics may be constructed if this procedure is repeated a few times, resulting in an overall average SNR value for the chosen setting, for instance. A step-by-step increase of the percentage of input images used for the construction of a light field allows to assemble a trend of important parameters, dependent on the number of input images.

To be more precisely, a first light field is constructed based on a subset of 5% of images randomly selected from the set of 341 input images. From this light field, 341 views are reconstructed, exactly corresponding to the positions of the 341 input images. An SNR value according to Equation 7.9 is then computed for each of the thus reconstructed views. In order to be able to derive a reasonable average SNR value this experiment using 5% of the input imagery is repeated five times, thereby always selecting a new random subset of images, of course. The thus derived SNR values allow to obtain a sense of the quality of reconstructed views achievable using the respective number of input images in the given setting.

<table>
<thead>
<tr>
<th>Example</th>
<th>Figure</th>
<th>RMSE Luminance</th>
<th>RMSE all channels</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example (1)</td>
<td>Figure 7.23</td>
<td>$25.41 &gt; \tau_l$</td>
<td>$45.89 &gt; \tau_a$</td>
<td>accepted</td>
</tr>
<tr>
<td>Example (2)</td>
<td>Figure 7.24</td>
<td>$13.47 &gt; \tau_l$</td>
<td>$24.44 &gt; \tau_a$</td>
<td>accepted</td>
</tr>
<tr>
<td>Example (3)</td>
<td>Figure 7.25</td>
<td>$21.75 &gt; \tau_l$</td>
<td>$39.59 &gt; \tau_a$</td>
<td>accepted</td>
</tr>
<tr>
<td>Example (4)</td>
<td>Figure 7.26</td>
<td>$10.57 &lt; \tau_l$</td>
<td>$18.95 &lt; \tau_a$</td>
<td>rejected</td>
</tr>
<tr>
<td>Example (5)</td>
<td>Figure 7.27</td>
<td>$10.50 &lt; \tau_l$</td>
<td>$18.96 &lt; \tau_a$</td>
<td>rejected</td>
</tr>
<tr>
<td>Example (6)</td>
<td>Figure 7.28</td>
<td>$11.41 &lt; \tau_l$</td>
<td>$20.55 &lt; \tau_a$</td>
<td>rejected</td>
</tr>
</tbody>
</table>
The above scenario is reiterated for 10–100% of random choices from the input image set with increments of 5%, resulting in 20 levels of different input set sizes the respective light fields are constructed from. Note that the random selection on each level is repeated five times, i.e. five different light fields per level with unequal random sets of input images are constructed, from each of which the 341 views are being reconstructed and compared with their corresponding reference images.

It is, however, important to note that the grid of the $xy$-parameter space with its resolution set to $32 \times 32$ does not coincide with the regular grid of the set of input images. Figure 7.29 gives an illustration of the two regular grids involved on top of each other in perfect match. The red dots represent the grid of $31 \times 11 = 341$ COPs of all input images whereas the black dots depict the sample locations in the $xy$-parameter space. Apart from the four corners, no red dot coincides with a black dot and vice versa. Since the linear basis functions employed are centered on each sample location of the black grid and thus perfect reconstructions are only possible at such a location (Section 7.4.1) the four corners are factored out of the analysis of SNR values because the signal-to-noise ratio becomes infinite for a signal and its perfect reconstruction (Equation 7.9).

The five series of 341 computed SNR values per level are averaged on each level separately and visualized together with the results of all 20 levels. The resulting charts are shown in Figure 7.30, revealing a linear evolution of the average SNR depending on the percentage of input images used for the construction of a light field (Figure 7.30 a). A comparison of the average curve with the evolution of the maximum and minimum curves is given in Figure 7.30 b).

Figure 7.30 clearly shows that the number of SNR values closer to the minimum value is larger than the ones close to the maximum values, i.e. the average curve lies much closer to the minimum than to the maximum curve. Obviously only few input images get reconstructed significantly above-average—namely the ones with their COP close to a sample location in the parameter space,—which clearly unveils the limited interpolation abilities of the quadra-linear basis functions in use.

### Table 7.9
Parameter set describing the example light fields used in Section 7.6.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Example light fields of Section 7.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light field's resolution</td>
<td>$32 \times 32 \times 128 \times 128$</td>
</tr>
<tr>
<td>Parameterization model</td>
<td>Sphere–plane</td>
</tr>
<tr>
<td>Angular spacing ($\Delta \varphi$, $\Delta \theta$)</td>
<td>$(2.9^\circ, 1.0^\circ)$</td>
</tr>
<tr>
<td>Set of input images used</td>
<td>Husky (Table B.4)</td>
</tr>
<tr>
<td>Number of input images used</td>
<td>$17 , (5%) - 341 , (100%), \text{ gradually } +5%$</td>
</tr>
<tr>
<td>Interpolation filter</td>
<td>Gauss, 5-tap</td>
</tr>
<tr>
<td>Wavelet basis</td>
<td>Haar</td>
</tr>
<tr>
<td>Maximum transformation level</td>
<td>$M = 4$</td>
</tr>
</tbody>
</table>
Regular evaluation grids used for the overall SNR measures shown in Figure 7.30, put on top of each other.

- Red points: Regular grid of input images.
- Black points: Regular grid of the light field parameterization.

Figure 7.31 a) additionally shows the evolution of the average distance between neighboring input camera locations, i.e. COPs. The distance is determined by searching for the closest neighbor for each camera location in the subset of used input images, using the angular distance in terms of sphere coordinates. Again, Figure 7.31 b) compares the average curve with the trends of the minimum and the maximum curve. Note that the data these charts are based on is generated and averaged in the same way as the SNR values are, i.e. the angular distances are computed for all five series on each of the 20 levels.

The discussed experiments visualize the correlation between the number of input images used and the average quality of reconstructed views in terms of signal-to-noise ratio, revealing the linear dependency in the given setting. These experiments may, however, be conducted in more than just one way. On the one hand, a test light field may be constructed from a subset of randomly chosen input images by completing the core's
7.6 Dependency on Filling Degree

FIGURE 7.30 Evolution of the overall SNR relative to the percentage of input images used. 

- a) Average of the 5 series on each level. 
- b) Maximum, minimum and average curve of the 5 series on each level.

FIGURE 7.31 Evolution of the angular spacing between neighboring input camera locations, relative to the percentage of input images used. 

- a) Average of the 5 series on each level. 
- b) Maximum, minimum and average curve of the 5 series on each level.

approximation mode. Following this approach, the test light field needs to be regenerated for each of the 20 levels.

On the other hand, a fixed test light field may first be generated based on just a few randomly chosen input images. The number of images the light field is built from may then be gradually increased through random selection of a set of new input images which are inserted using the local update procedure. Of course, this selection must be repeated for all 20 levels. Moreover, the images used for the construction of the fixed base light field must be excluded from any subsequent selection, preventing them from being used more than once. Note that all reconstructed views needed to determine the SNR values are computed using the local reconstruction procedure, independent on the strategy chosen.

It is, however, important to note that both scenarios will produce comparable results from a qualitative point of view as is shown in Figure 7.30 a) but with a weak difference from a quantitative point of view due to the non-linear element during the construction of the eMRA (Section 4.6).

After all, the experiments performed and described follow the first strategy due to the following reason. Having an acquisition application using a hand-held video camera in mind, the experiment was conducted in order to find an answer to the question of how many input images should be acquired for the completion of the approximation mode, i.e. what is the optimal moment to switch to local update operations. Facing the linear result
in Figure 7.30 a) there is no such point. Hence, the overall quality of a light field after having completed the approximation mode may be set very individually—every additional input image used will equally improve the estimate, in case the images are more or less evenly distributed.

This linear correlation offers the possibility to take (1) the object’s properties and (2) the target quality of the light field into account, while acquiring input imagery.

Firstly, if an object’s surface runs very smooth it is the simplest to acquire an evenly distributed set of input images and construct the light field from it. If, in turn, the object shows some very detailed but small areas, the basic set just needs to be dense enough to satisfy the overall quality of reconstructed views since the details may be captured thereafter using the local update procedure during the oracle mode—that is, the initial set does not need to be as densely sampled as it would be required by the small detailed areas.

Secondly, if the interest is put on an overall reconstruction quality that is as good as possible it is again the best strategy to acquire the possibly sizable set of input images that satisfies the target quality and construct the light field from it. If the main interest lies in a nice reconstruction of small parts of an object, the initial light field may be kept small by acquiring just a handful of images for the first estimate since the region of interest may be specifically sampled afterwards, throughout the oracle mode.

### 7.7 IMPACT OF COMPRESSION

Clearly, any lossy compression of the underlying data representation will impose some implication on the quality of reconstructed views. The following examples contrast the achievable compression ratios with the reconstruction quality, based on a light field built using real-world imagery, i.e. the Haathi scene (Table B.6 on page 172). Moreover, the different attainable compression ratios of (1) the in-core compression, based on the structure introduced in Section 6.2.3, (2) the data written to disk, using the file format discussed in Section 6.2.4 and finally (3) the stored file which is further compressed are given and commented on.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Example light fields of Section 7.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light field’s resolution</td>
<td>$16 \times 16 \times 256 \times 256$</td>
</tr>
<tr>
<td>Parameterization model</td>
<td>Sphere–plane</td>
</tr>
<tr>
<td>Angular spacing ($\Delta \phi$, $\Delta \phi^*$)</td>
<td>$(2.0^\circ, 2.0^\circ)$</td>
</tr>
<tr>
<td>Set of input images used</td>
<td>Haathi (Table B.6)</td>
</tr>
<tr>
<td>Number of input images used</td>
<td>256</td>
</tr>
<tr>
<td>Interpolation filter</td>
<td>Gauss, 5-tap</td>
</tr>
<tr>
<td>Wavelet basis</td>
<td>Haar, Daubechies–4</td>
</tr>
<tr>
<td>Maximum transformation level</td>
<td>$m = 1, M = 4$</td>
</tr>
</tbody>
</table>
7.7 IMPACT OF COMPRESSION

The test light field with its resolution set to $16 \times 16 \times 256 \times 256$ is constructed from 256 input images, randomly selected from the pool of available images of the Haathi scene. All results shown in this Section 7.7 were computed by using either (1) the Haar or the (2) Daubechies–4 wavelet basis during the approximation mode of the test light field. The transformations to wavelet space were completed to the first level $m = 1$ or the maximum possible level $m = M$, resulting in quite different compression ratios as is shown in the following illustrations. Again, the complete set of parameters the test light fields are based on are compiled in Table 7.10.

Figure 7.32 visualizes the impact of the in-core compression on the quality of a reconstructed view from the test light field, transformed to the first level using the Haar basis. For comparison, image a) shows a rendering from the untransformed light field. Image b) right next to it shows the same reconstruction from the transformed but uncompressed data set, i.e. all signal energy $E_I$, computed according to Equation 7.5, is preserved. As is expected, images a) and b) are identical.

Images c)–k) show reconstructions of again the same view but from compressed light fields that preserve different amounts of signal energy, i.e. from 99% to 91%. Note that once coefficients from the lowest-frequency subband get eliminated, serious artifacts, i.e. holes, occur in the reconstructions (Section 4.3.6). This obviously happens for $E_{I_{compr}} \leq 0.93 \cdot E_I$. Beforehand, less serious artifacts are visible due to neglected detail coefficients. However, a compression up to $0.94 \cdot E_I$ is visually acceptable.

Naturally, the elimination of coefficients from the DC part of a signal, i.e. coefficients from the lowest-frequency subband, is highly undesirable due to these serious reconstruction errors. Therefore, if not provoked for illustration purposes as is done in Figures 7-32 through to 7.35, coefficients from the DC portion are normally left untouched.

Figure 7.33 illustrates the impact of the in-core compression on the quality of the same reconstructed view with the light field again transformed to the first level but using the Daubechies–4 wavelet. Image a) again shows a reconstruction from the untransformed data set whereas image b) shows a rendering from the transformed but uncompressed data set. However, images c)–k) illustrate the same artifacts due to the same reasons as is the case in Figure 7.32.

Contrary to the examples given before, Figures 7.34 and 7.35 show the same reconstructed view but computed from a light field transformed to the maximum possible level $m = M$, using the Haar wavelet (Figure 7.34) and the Daubechies–4 basis (Figure 7.35).

Again note the serious image degradation caused for these illustrations once coefficients from the DC subband get eliminated by the compression procedure. In contrast to transforms to the first level $m = 1$ such artifacts occur for $E_{I_{compr}} \leq 0.75 \cdot E_I$. Also note the different artifacts for $E_{I_{compr}} \geq 0.80 \cdot E_I$ caused by the wavelet basis chosen, e.g. the well-known blocking artifacts of the Haar wavelet in Figure 7.34.

Figure 7.36 on page 159 illustrates the decrease regarding the SNR measure computed according to Equation 7.9 for the image series given before. Chart a) visualizes the trends for the transforms to the first level shown in Figures 7.32 and 7.33. A compression preserving $0.94 \cdot E_I$ proved to yield acceptable results regarding visual quality. Chart b) of Figure 7.36, on the other hand, gives the trends for the transforms to the maximum possible level shown in Figures 7.34 and 7.35. For this latter case, compressions down to $0.80 \cdot E_I$ are acceptable in terms of visual quality. Note that these different limits for satisfying compression ratios result in comparable SNR values in either cases.
FIGURE 7.32 Impact of compression on the visual quality of a positionally fixed, reconstructed view from a level-1 transformed data set using the Haar wavelet.

a) Rendering from untransformed data.
b) Rendering from transformed data, preserving all signal energy.
c)–k) Renderings from transformed data, preserving different amounts of signal energy.
FIGURE 7.33 Impact of compression on the visual quality of a positionally fixed, reconstructed view from a level-1 transformed data set using the Daubechies-4 wavelet.

a) Rendering from untransformed data.
b) Rendering from transformed data, preserving all signal energy.
c)–k) Renderings from transformed data, preserving different amounts of signal energy.
FIGURE 7.34  Impact of compression on the visual quality of a reconstructed view, rendered from a transformed data set to the maximal possible level using the Haar wavelet.

FIGURE 7.35  Impact of compression on the visual quality of a reconstructed view, rendered from a transformed data set to the maximal possible level using the Daubechies–4 wavelet.
7.7 IMPACT OF COMPRESSION

**Figure 7.36** Comparison of the SNR trends for the image series shown in Figures 7.32–7.35.
- a) Comparison between level-1 transforms using Daubechies-4 and Haar wavelets.
- b) Comparison between transforms to the maximal level, using the same two wavelets.

Figure 7.37 below additionally gives the achievable in-core compression ratios for the previous image series. Contrary to the SNR comparison, the different compression limits for a transform to the first level or to the maximum possible level differ significantly in terms of in-core compression ratios. That is, a compression of 0.94 \( \cdot E_t \) for a level-1 transformed light field using either of the used wavelet basis yields a ratio of about 1:4.5 whereas the transform to the maximum possible level allows for a ratio of about 1:6 for 0.80 \( \cdot E_t \)—which comes at no surprise since the deeper a transform pyramid is the less coefficients capture a signal’s energy on the lowest-frequency subband. Hence, more detail coefficients may be eliminated before serious artifacts, e.g. openings, in the signal’s reconstruction occur.

**Figure 7.38** Confronts the compression factors of (1) the in-core compression, (2) the compressed data set written to file and (3) the stored file that is further compressed for both wavelet bases and both scenarios used for the previous image series. Note that using the Daubechies-4 wavelet, for instance, a compression ratio of nearly 1:190 is feasible,
without requantization. As a consequence, the additional computational costs for the transformation to the maximum possible level—which are dominated by the 1-level transform anyway (Section 5.4)—do pay off regarding the compression ratios achievable as was discussed before in Section 6.2.5. However, an in-depth discussion of the compression module regarding (1) the computational performance and (2) the compression factors winnable was already given in Section 6.2.5.

Figure 7.39 finally illustrates the compression factors given in Figure 7.38 below in terms of actual memory needs. The horizontal line in all four charts corresponds to the constant memory needs if the light field data was stored in a flat array structure.

**FIGURE 7.38** Comparison of compression factors in-core, when stored using the custom file format and when additionally compressing the written files, based on the image series shown in Figures 7.32–7.35.

a) Comparison of level-1 transforms using the Haar wavelet.

b) Comparison of level-1 transforms using the Daubechies–4 wavelet.

c) Comparison of transforms to the maximal level, using the Haar wavelet.

d) Comparison of transforms to the maximal level, using the Daubechies–4 wavelet.
FIGURE 7.39 Comparison of memory consumption of a flat array, the in-core compression, the custom file format and the further compressed written file, based on the image series shown in Figures 7.32–7.35.

a) Comparison of level-1 transforms using the Haar wavelet.

b) Comparison of level-1 transforms using the Daubechies-4 wavelet.

c) Comparison of transforms to the maximal level, using the Haar wavelet.

d) Comparison of transforms to the maximal level, using the Daubechies-4 wavelet.
Seite Leer /
Blank leaf
After the presentations of evaluations concerning representation, operators and concepts given in the previous Chapter 7 the following Section 8.1 summarizes the results and contributions of this thesis. A short discussion of findings and conclusions in Section 8.2 complements on this summary, followed by a short list of directions for future work in Section 8.3.

8.1 SUMMARY

The principal solutions developed in the course of this thesis along with their benefits are compiled and briefly commented in the list given below.

**Representation & Update scheme.** Chapter 4 introduced the extended multiresolution analysis (eMRA) as a hierarchical representation for light field data. It combines the benefits of a hierarchical scattered data interpolation method with wavelet coding. Together with the oracle mechanism which is based on the incremental update scheme the eMRA representation forms an integrated concept for progressive acquisition and processing of light field data.

**Local operators.** Chapter 5 introduced a set of efficient operators that directly connect to the eMRA representation. A local projection operator allows for progressive refinement of the initial representation using the incremental update scheme whereas a local reconstruction operator enables rendering from the wavelet-transformed data. Being local operators, none of them requires an expensive complete inverse transform. Computational performance analyses confirm the effectiveness of the concept of only locally working operators. Additionally, various examples demonstrate the visual performance of the incremental update procedure in combination with the insertion operator.

**Compression scheme.** A lossy multi-stage compression scheme based on wavelet coefficient thresholding shows the feasibility of efficient in-core compression in the highly dynamic setting while still fully supporting both local operators. The scheme fea-
tures respectable compression ratios in-core and very competitive compression factors out-of-core, even without any requantization of coefficients.

**System integration.** An implementation based on a strictly modular design nicely integrates all previously mentioned concepts in a system with transparent interaction as was indicated with the overview given in Chapter 3. This system offers free configuration of fundamental modules at run time, such as the choice of parameterization method the respective module should apply, i.e. the cylinder–plane or the sphere–plane setting, or the choice of memory model, i.e. an uncompressed and thus faster representation for smaller examples or the compressed representation as it was presented in Chapter 6.

**Evaluation.** The system was tested in parts and as a whole using various sets of synthetic pre-computed images. Chapter 7 finally comments on a number of experiments based on real-world input material read from a video camera. These tests focus firstly on parameters that reflect global improvement tendencies, secondly on the effects of the incremental update scheme regarding local improvement and selectivity and thirdly on the impact of compression on the visual quality.

### 8.2 CONCLUSIONS

Given the various analyses in the thesis and the results summarized in the previous section in particular there are a few conclusions to arrive at. These are discussed below, broken down into two sections dealing with the suitability of the approaches and their limitations learned.

#### 8.2.1 Suitability

The usage of wavelet theory in the present setting proved to be very valuable. Many of the wavelet's properties are taken advantage of, resulting in a flexible and highly integrated representation that shows many features, accordingly.

Firstly, the inherent hierarchy of a wavelet transform optimally matches with the hierarchical scattered data interpolation scheme. The hierarchy that needs to be computed anywise during the HSDI process is further exploited by commuting it into a consistent MRA representation. Thus, the hierarchy itself as well as its generation form an integral part of the data acquisition process.

Being based on wavelet theory, the eMRA representation enables the usage of all other properties of wavelets, such as the spatial localization, for instance. This feature permits the construction of highly efficient local operators absolutely needed in a dynamic setting.

Furthermore, since compression is an important issue in the field of IBR, the integration of a wavelet–based compression scheme is obvious. The design of a suitable data structure even allows for a dynamic in-core compression of the eMRA representation.

Consequently, wavelets proved to be very well suited not only for the purpose of hierarchically representing multi-dimensional data but also for processing it. Moreover, wavelet theory enabled the design of the unified representation for acquisition, processing, rendering and compression of light field data.
8.2.2 Limitations

As a matter of course, the solutions chosen also exhibit limitations. Most importantly, the processing of wavelet–transformed multi–dimensional and multi–channel data shows a substantial computational complexity. Though edit operations, i.e. image insertions just as image reconstructions, too, may be processed at interactive rates on single CPU computer systems, this is true for comparably smallish images of resolution of about 64 × 64. Processing times change considerably for higher resolutions.

Moreover, the usage of the quadra–linear basis functions demands to involve not just one but four 2–dimensional slices of the 4D data set, i.e. images, in the general case, as was discussed in Section 6.1.3. However, using higher order and smoother basis functions would be desirable, mainly in order to overcome the limited interpolation capabilities of the quadra–linear basis functions in use. Clearly, using wider kernels automatically means even more computational load since more 2D slices must be involved per local operation.

The operators based on slicing, however, impose an other severe limitation on the camera's freedom of movement, as was also mentioned in Section 6.1.3. Camera positions off the surface of the xy–parameter space require some other model for the computation of local insertions and/or local reconstruction operations. That is, the implementation of local operators based on slicing does not work for such camera positions. Pre–warping new input images before processing them using the local update operator and vice versa in the case of rendering, i.e. post–warping locally reconstructed views, would weaken the restriction but cannot completely solve the underlying problem.

As a consequence, there is still room for relevant improvements in the field of the parameterization, the local operators and their interaction. However, the above limitations do not disable the eMRA representation, nor its operator setting or parts of it.

8.3 DIRECTIONS FOR FUTURE WORK

Beside the above mentioned restrictions worth of being tackled there are several issues more that might be addressed in the future as is discussed below.

8.3.1 Engineering Extensions

The integration of the processing scheme as was developed in the course of this thesis in an easy–to–handle and user–friendly light field acquisition application based on a simple hand–held video camera—the idea that originally inspired this thesis—requires to solve for a few engineering questions. Most importantly, a broad and thorough evaluation of available tracking systems becomes necessary. A suitable system must not only provide accurate positional information but also the camera's orientation, i.e. the system must be able to track the camera's complete pose.

Moreover, since the update rate of a decent tracking system will most probably not match the frame rate of the video camera in use a solution in order to reliably and properly associate measured positional data to individual video frames becomes unavoidable.

As soon as a user captures video sequences of an object using a hand–held video camera she/he will not be able to supervise the acquisition process on a computer monitor. Thus, completely new ways of providing feedback and guiding the user must be developed. A nice idea would be an information loop–back mechanism from the acquisition system to
the video camera operated by the user, such as false-color-coded images with other information superimposed, for instance. Unfortunately, nowadays consumer cameras have no functionality to display external video and/or image data on their built-in LCD screen while recording.

Additionally, there is a great variety of minor system parts that would also needed to be made more practicable. The RMS threshold value needed for the selective update scheme during the oracle mode, for instance, cannot be specified by the user alone. An arguable solution would be the acquisition module learning a proper threshold value based on a few user decisions. Another example for possible improvement regarding more facile usability might be a more automated switch of modes of the system’s core component that would not need any user interaction anymore (Section 3.3.3). Instead of the analysis of difference image information only other indicators might be involved, too, such as the distribution of image samples recorded so far, for instance.

8.3.2 Algorithmic Extensions

Apart from topics concerning system engineering there is also room for improvements on the algorithmic and conceptional side, of course. The limitations regarding the basis functions and the restrictions on the camera’s freedom of movement mentioned in Section 8.2 hint at two obvious issues.

An interesting and challenging extension from an algorithmic point of view would be a generalization of the concept of local operators. That is, relaxing the constraint of only processing axis-aligned planes in high-dimensional data sets to more general subspaces of any dimensionality—up to the dimensionality of the data set itself, of course—would widen the range of possible applications of the concept of local operators. Moreover, it might also give a solution to the restriction on the camera’s freedom of movement by enabling the local editing of ray-based queries again.

However, there are many more areas of application for a unified data representation in combination with highly integrated editing operators and compression capabilities of virtually any dimensionality. Similar ideas that give evidence for the need of such flexible representations may be found in the field of information and data visualization [30, 51], for instance, as was already touched in Section 5.1.6 on page 92.

After all, based on the results of this thesis, I personally believe that a future development towards such a configurable, unified and highly integrated representation including an efficient local operator framework and an effective compression scheme for any kind of data would be the most interesting and most promising direction for future work to be followed. Clearly, the focus would then move away from the field of image-based rendering and modeling methods in particular—a trend that is, by the way, also noticeable by means of the ongoing evolution of computer graphics away from purely geometry-based approaches and away from purely image-based approaches, i.e. LF-like methods, towards hybrid systems. Moreover, an abstract wavelet-based representation and a suitable operator framework even aims beyond computer graphics in general.
The following partial wavelet transform (PWT) algorithm is the core routine to all wavelet transform procedures in the system. It applies an arbitrary wavelet filter to a data sequence \( a[] \) of size \( n \) for \( \text{isign} \) set to 1 or its transpose for \( \text{isign} \) set to -1. The actual wavelet filter employed is determined by previously setting the number of its coefficients (ncof), its arrays of coefficients (cc and cr) and the offset values (ioff and joff).

The PWT algorithm follows the wrap-around strategy by appropriately adjusting the pointers into the data sequence. The routine as is shown below has been adapted and extended from its original version printed in [92].

```c
void cLFO_Filter_wavelet::PWT(float a[], unsigned long n, int isign)
//partial wavelet transform, working on float values
{
    float ai, ail;
    unsigned long i,ii,j,jf,jr,k,n1,ni,nj,nh,nmod;
    if (n<4) return;
    if (wksp_sz<n+1) { //check workspace
        if (wksp_flt!=0) {
            delete[] wksp_flt;
            wksp_flt = 0;
        }
        wksp_sz = n+1;
        wksp_flt = new float[wksp_sz];
    }
    nmod=ncof*n; //a positive constant equal to zero mod n
    n1=n-1; //mask of all bits, since n a power of 2
    nh=n>>1;
    memset(wksp_flt,0,wksp_sz*sizeof(float));
}```
if (isign >= 0) {
    // apply filter:
    for (ii = 1, i = 1; i <= n; i += 2, ii++) {
        ni = i + nmod + ioff; // pointer to be incremented and...
        nj = i + nmod + joff; // wrapped-around
        for (k = 1; k <= ncof; k++) {
            nf = nl & (ni + k); // bitwise AND to wrap-around...
            nj = nl & (nj + k); // the pointers
            wksp_flt[ii] += cc[k] * a[f];
            wksp_flt[ii + nh] += cr[k] * a[jr + 1];
        }
    }
}
else {
    // apply transposed filter:
    for (ii = 1, i = 1; i <= n; i += 2, ii++) {
        ai = a[ii];
        ail = a[ii + nh];
        ni = i + nmod + ioff;
        nj = i + nmod + joff;
        for (k = 1; k <= ncof; k++) {
            nf = (nl & (ni + k)) + 1;
            nj = (nl & (nj + k)) + 1;
            wksp_flt[jf] += cc[k] * ai;
            wksp_flt[jr] += cr[k] * aill;
        }
    }

    for (j = 1; j <= n; j++) a[j] = wksp_flt[j]; // copy back
The following Tables B.1 through B.6 list the complete parameter sets of all lots of input images used for the generation of example light fields and other illustrations shown in this thesis.

Note that the parameters summarized do cover the input image sets only. The parameters that exactly describe therewith generated light field data are given in previous chapters, along with the illustration of the respective example light field.

### TABLE B.1 Parameter set describing the input images of the Truck scene, used for the light field shown in Figure 4.8 on page 80 (Truck model |ALIAS©Models|).

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Truck scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>150</td>
</tr>
<tr>
<td>Images’ resolution</td>
<td>256 x 256</td>
</tr>
<tr>
<td>Image grid (horizontally x vertically)</td>
<td>30 x 5</td>
</tr>
<tr>
<td>Image distribution</td>
<td>cylindrical</td>
</tr>
<tr>
<td>Image source</td>
<td>synthetic (raytraced)</td>
</tr>
<tr>
<td>Angular spacing (horizontally)</td>
<td>6°</td>
</tr>
<tr>
<td>Angular range (horizontally)</td>
<td>180°</td>
</tr>
</tbody>
</table>
### TABLE B.2
Parameter set describing the input images of the **Falls** scene, used for the light field shown in Figure 4.9 on page 82 (Falls model [ALIAS©Models]).

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Falls scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>450</td>
</tr>
<tr>
<td>Images' resolution</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Image grid (horizontally x vertically)</td>
<td>45 × 10</td>
</tr>
<tr>
<td>Image distribution</td>
<td>cylindrical</td>
</tr>
<tr>
<td>Image source</td>
<td>synthetic (raytraced)</td>
</tr>
<tr>
<td>Angular spacing (horizontally only)</td>
<td>1°</td>
</tr>
<tr>
<td>Angular range (horizontally only)</td>
<td>45°</td>
</tr>
</tbody>
</table>

### TABLE B.3
Parameter set describing the input images of the **Frog** scene, used for the light field shown in Figure 7.8 on page 130.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Frog scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>91</td>
</tr>
<tr>
<td>Images' resolution</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Image grid (horizontally x vertically)</td>
<td>13 × 7</td>
</tr>
<tr>
<td>Image distribution</td>
<td>spherical</td>
</tr>
<tr>
<td>Image source</td>
<td>real (captured from video stream)</td>
</tr>
<tr>
<td>Angular spacing, horizontally</td>
<td>5°</td>
</tr>
<tr>
<td>Angular spacing, vertically</td>
<td>5°</td>
</tr>
<tr>
<td>Angular range, horizontally</td>
<td>$\phi = [-30^\circ, ..., 30^\circ]$</td>
</tr>
<tr>
<td>Angular range, vertically</td>
<td>$\psi = [0^\circ, ..., 30^\circ]$</td>
</tr>
</tbody>
</table>
TABLE B.4  Parameter set describing the input images of the Husky scene, used for the light field shown in Figure 7.9 on page 131.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Husky scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>341</td>
</tr>
<tr>
<td>Images' resolution</td>
<td>256 x 256</td>
</tr>
<tr>
<td>Image grid (horizontally x vertically)</td>
<td>31 x 11</td>
</tr>
<tr>
<td>Image source</td>
<td>real (captured from video stream)</td>
</tr>
<tr>
<td>Angular spacing, horizontally</td>
<td>3°</td>
</tr>
<tr>
<td>Angular spacing, vertically</td>
<td>3°</td>
</tr>
<tr>
<td>Angular range, horizontally</td>
<td>$\phi = [0°, ..., 90°]$</td>
</tr>
<tr>
<td>Angular range, vertically</td>
<td>$\theta = [30°, ..., 60°]$</td>
</tr>
</tbody>
</table>

TABLE B.5  Parameter set describing the input images of the Ganesha scene, used for the light field shown in Figure 7.10 on page 131.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Ganesha scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>256</td>
</tr>
<tr>
<td>Images' resolution</td>
<td>256 x 256</td>
</tr>
<tr>
<td>Image grid (horizontally x vertically)</td>
<td>16 x 16</td>
</tr>
<tr>
<td>Image distribution</td>
<td>spherical</td>
</tr>
<tr>
<td>Image source</td>
<td>real (captured from video stream)</td>
</tr>
<tr>
<td>Angular spacing, horizontally</td>
<td>2°</td>
</tr>
<tr>
<td>Angular spacing, vertically</td>
<td>2°</td>
</tr>
<tr>
<td>Angular range, horizontally</td>
<td>$\phi = [0°, ..., 30°]$</td>
</tr>
<tr>
<td>Angular range, vertically</td>
<td>$\theta = [10°, ..., 40°]$</td>
</tr>
</tbody>
</table>
TABLE B.6 Parameter set describing the input images of the *Haathi scene*, used for the light field shown in Figure 7.11 on page 131.

<table>
<thead>
<tr>
<th>Type of parameter</th>
<th>Haathi scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of images</td>
<td>1024</td>
</tr>
<tr>
<td>Images' resolution</td>
<td>256 × 256</td>
</tr>
<tr>
<td>Image grid (horizontally x vertically)</td>
<td>32 × 32</td>
</tr>
<tr>
<td>Image distribution</td>
<td>spherical</td>
</tr>
<tr>
<td>Image source</td>
<td>real (captured from video stream)</td>
</tr>
<tr>
<td>Angular spacing, horizontally</td>
<td>1°</td>
</tr>
<tr>
<td>Angular spacing, vertically</td>
<td>1°</td>
</tr>
<tr>
<td>Angular range, horizontally</td>
<td>( \varphi = [0^\circ, \ldots, 31^\circ] )</td>
</tr>
<tr>
<td>Angular range, vertically</td>
<td>( \theta = [10^\circ, \ldots, 41^\circ] )</td>
</tr>
</tbody>
</table>
C.1 NOMENCLATURE

CHAPTER 2 IMAGE-BASED METHODS

\( P_{lf} \) : Plenoptic function.

\( L(s, t, u, v) \) : Light field or Lumigraph function, continuous, 4-dimensional.

\( L_{proj}(s, t, u, v) \) : Light field or Lumigraph function, projected onto some basis.

\( B_{i, j, p, q} \) : Basis function, 4-dimensional.

\( x_{i, j, p, q} \) : Coefficients associated with basis functions \( B_{i, j, p, q} \).

\( \tilde{B}_{i, j, p, q} \) : Duals of the basis function \( B_{i, j, p, q} \).

\( \tilde{x}_{i, j, p, q} \) : Coefficients associated with dual basis functions \( \tilde{B}_{i, j, p, q} \).

\( U \) : Parameter space.

\( T \) : Attribute space.

\( S(U) \) : Support function, mapping from parameter space to ray space.

\( A(U) \) : Attribute function, mapping from parameter to attribute space.

\( m \) : Hierarchy level.

\( \psi_m \) : Wavelet function on level \( m \).
CHAPTER 4  DATA REPRESENTATION

**R** .......... Set of real numbers.

**Z** .......... Set of integers.

**N** .......... Set of positive integers, excluding zero.

**N₀** .......... Set of positive integers, including zero.

$L^2(R)$ .......... Space of square-integrable functions.

$l^2(Z)$ .......... Space of square-summable sequences.

$$\langle f | g \rangle \quad \text{Inner product of functions } f \text{ and } g, \text{ i.e. } \int_{-\infty}^{+\infty} f(x)g(x)dx.$$ 

$\delta_{ab} \quad \text{Kronecker delta function, i.e. } \delta_{ab} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{else} \end{cases}$

$I$ .......... Identity matrix.

$m$ .......... Current approximation level.

$M$ .......... Maximum approximation level.

$\phi(x)$ .......... Scaling function (mother function).

$\phi_{mk}(x)$ .......... Scaling function (dilation $m$, shift $k$).

$V_m$ .......... Subspace of $L^2(R)$, spanned by $\phi_{mk}(x)$.

$c_{mk}$ .......... Scaling coefficient, a scalar value corresponding to $\phi_{mk}(x)$.

$\psi(x)$ .......... Wavelet function (mother function).

$\psi_{mk}(x)$ .......... Wavelet function (dilation $m$, shift $k$).

$W_m$ .......... Subspace of $L^2(R)$, spanned by $\psi_{mk}(x)$.

$d_{mk}$ .......... Wavelet coefficient, a scalar value corresponding to $\psi_{mk}(x)$.

$\text{supp}(g)$ .......... Support of function $g(x)$.

$\Delta_x^f$ .......... Localization of function $f(x)$ in the spatial domain.

$\Delta_{\tilde{f}}$ .......... Localization of function $\tilde{f}(x)$ in the frequency domain.

$F(\omega)$ .......... Fourier transform of function $f(x)$.

$\tilde{\phi}(x)$ .......... Dual scaling function.

$\tilde{\psi}(x)$ .......... Dual wavelet function.
**CHAPTER 5 LOCAL OPERATORS**

$h$ ............... Filter sequence, used for the FWT.

$s_h$ ............... Size of filter $h$.

$j_h$ ............... Current center position of filter $h$ in a data set.

$j_{h,m}$ ............... Current center position of $j_h$ on approximation level $m$.

$A(m, h, s_h, j_h)$ ........ Active area on level $m$.

$r_c$ ............... Cache radius.

$c_{ID}$ ............... Virtual computational cost for filtering of a 1D data field.

**CHAPTER 7 EVALUATION**

$p_i$ ............... $i^{th}$ pixel value of one channel of some image.
\( N \) ................. Total number of pixels in some image.
\( E_{\text{continuous}} \) .......... Signal energy of some continuous signal.
\( E_{\text{discrete}} \) .......... Signal energy of some discrete signal.
\( E_i \) .................. Energy of some multi–channel image.
\( E_{p} \) ................. Energy of a single pixel of some multi–channel image.
\( E_{l} \) .................. Energy of wavelet–transformed, multi–channel light field data.

\( \tau_{l} \) .................. Update threshold, applied to luminance–based RMSE value.
\( \tau_{a} \) .................. Update threshold, applied to all–color–channel RMSE value.

### C.2 ABBREVIATIONS

2PP ............... 2–plane parameterization (2 points on 2 planes) [16].
2SP ............... 2–sphere parameterization (2 points on 2 spheres) [16].

A
AAL ................. Adaptive acquisition of lumigraphs from synthetic scenes [107].
AC ................. Alternating current (electrical engineering context, see also DC).

B
BRDF ............... Bi–directional reflectance distribution function [65].

C
CCD ................. Charge coupled device.
CMOS ............... Complementary metal oxide semiconductor.
COP ............... Center of projection.

D
DC ................. Direct current (electrical engineering context, see also AC).
DCT ............... Discrete cosine transform.
DPP ............... Direction and point parameterization [16].

E
eMRA ............... Extended multiresolution analysis.

F
FPGA ............... Field–programmable gate array.
FWT ............... Fast wavelet transform.
H
HPF .......... Hierarchical polynomial fit filtering [14].
HSDI .......... Hierarchical scattered data interpolator/-ion.
I
IBM ............ Image—based modeling.
IBMR ............ Image—based modeling and rendering.
IBR ............ Image—based rendering.
IEEE .......... Institute of electrical and electronics engineers.
J
JIT ............ Just—in—time (with respect to rendering performance, i.e. real—time rendering) [139].
L
LCD ............ Liquid crystal display.
LF ............ Light field.
LFR ........... Light field rendering [75].
LUM ........... Lumigraph [45].
M
MPEG .......... Moving picture experts group.
MRA .......... Multiresolution analysis.
MSE .......... Mean square error (see also RMS, RMSE).
P
PDP .......... Point and direction parameterization [16].
PIWS .......... Progressive Inverse Wavelet Synthesis [139].
PLF .......... Plenoptic function [2].
PMIS .......... Plenoptic modeling and rendering from image sequences taken by a hand—held camera [56].
PS .......... Plenoptic Sampling [20].
PSNR .......... Peak signal—to—noise ratio (see also SNR).
PWT .......... Partial wavelet transform.
R
RBC .......... Reference block coder [145].
RMS .......... Root mean square (Square root of the mean of squared values).
RMSE .......... Root mean square error.
S
SDL ............... Scene description language [ALIAS©Tools].
SLF ............... Surface light fields for 3D photography [138].
SNR ............... Signal–to–noise ratio (see also PSNR).
SPIHT ............. Set Partitioning in Hierarchical Trees [106].
SPP ............... Sphere–plane parameterization [17].
STL ............... Standard template library [122].
SVQ ............... Spatial vector quantization.

U
ULR ............... Unstructured lumigraph rendering [12].

V
VDTM ............. View–dependent texture mapping [35, 36, 102].
VQ ............... Vector quantization [42].

X
XGA ............... Extended graphics adapter/array (1024 × 768 pixels).
REFERENCES


183


Seite Leer / Blank leaf
APPENDIX

COPYRIGHTS

E.1 SOFTWARE COPYRIGHTS


[ALIAS©Models] . . All models and demo scenes © 2000 Alias|Wavefront.

E.2 PICTURE COPYRIGHTS

[©Gortler] . . . . . Image data courtesy of Steven J. Gortler.
    sjg@cs.harvard.edu

[©Kaidan] . . . . . Image data courtesy of Kaidan Incorporated, Feasterville, PA, USA.
    Jim Anders, President of Kaidan.
    http://www.kaidan.com/

[©Levoy] . . . . . Image data courtesy of Marc Levoy.
    levoy@cs.stanford.edu

[©Naimark] . . . . Image data courtesy of Michael Naimark.
    http://www.naimark.net/
CURRICULUM VITAE

F.1 PERSONAL DETAILS

Name (first, last) . . . Reto Lütolf.
Place of birth . . . . Lucerne, Switzerland.
Nationality . . . . . Swiss.

F.2 EDUCATION

1998 — 2003 . . . . Research and teaching assistant,
Computer Graphics Laboratory, headed by Prof. Markus Gross,
Department of Computer Science,
Swiss Federal Institute of Technology ETH, Zurich, Switzerland.

1992 — 1998 . . . . Diploma degree in Computer Science,
Dipl. Informatik–Ing. ETH,
Swiss Federal Institute of Technology ETH, Zurich, Switzerland.

1985 — 1992 . . . . Matura Typus C,
Kantonsschule Alpenquai, Lucerne, Switzerland.
"Major e longinquo reverentia"

— Publius Cornelius Tacitus [127]