GRAPHICAL MODELS AND SIMULATION FOR THZ-IMAGING

GRAPHISCHE MODELLE UND SIMULATION FÜR THZ-BILDGEBUNG

DISSERTATION
zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften

von
Martin Pätzold

eingereicht bei der
Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen

Juni 2017

1. Gutachter: Prof. Dr. Andreas Kolb
2. Gutachter: Prof. Dr. Peter Haring Bolívar
Tag der mündlichen Prüfung: 24.01.2018

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier

ABSTRACT

The utilization of terahertz (THz) radiation is an active research field. Approaches for non-intrusive material monitoring or reliable surveillance of hidden treats promise considerable improvements of current technologies in these research fields. Due to the prototypical nature of current methods, easily accessible or unoptimized techniques are commonly used for processing THz data or simulating THz radiation. Methods of computer graphics (CG) allow an efficient prototyping to evaluate the potential of these new THz technologies and algorithms.

In this thesis, CG methods are developed and applied to the problems of THz research. It is shown that these methods are beneficial in terms of efficiency and performance. A sparse voxel tree (SVT) is proposed to store highly detailed objects with inner structures for simulating the imaging capabilities of THz setups. Mainly, an efficient creation and rendering of this voxel structure are introduced. The SVT is applied to the simulation of a prototypical THz setup which scans a scene by focused radiation.

Another THz setup, which is based on unfocused radiation and synthetic aperture techniques, is used to show the benefits of GPU algorithms for the reconstruction of large THz data. Furthermore, a volume based simulation of this system is shown. In addition, a geometrical configuration of the THz system and the rendering of multimodal data from the THz setup are introduced.

ZUSAMMENFASSUNG

Die Nutzbarmachung von Terahertz (THz) Strahlung ist ein aktives Forschungsfeld. Ansätze für berührungsfreie Materialüberwachung oder zuverlässliche Kontrolle von versteckten Bedrohungen versprechen erhebliche Verbesserungen von derzeitigen Technologien in diesen Forschungsfeldern. Durch den prototypischen Charakter aktueller Methoden werden leicht zugängliche oder unoptimierte Techniken bevorzugt um THz Daten zu verarbeiten oder THz-Strahlung zu simulieren. Methoden der Computergraphik (CG) erlauben ein effizientes Prototyping um das Potential von neuen THz-Technologien and Algorithmen zu evaluieren.

In dieser Arbeit werden CG-Methoden entwickelt und auf die Problemstellungen in der THz Forschung angewendet. Es wird gezeigt, dass diese Methoden in Bezug auf Effizienz und Performanz vorteilhaft sind. Ein Sparse Voxel Tree (SVT) wird vorgeschlagen um hochdetaillierte Objekte mit inneren Strukturen zu repräsentieren und die bildgebenden Möglichkeiten eines THz-Setups zu simulieren. Hauptsächlich wird die effiziente Erstellung und das Rendering dieser Voxelstruktur vorgestellt. Der SVT wird für die Simulation eine prototypischen THz-Setups eingesetzt, welches die Szene mit fokussierter Strahlung abtastet.

Ein anderes THz-Setup, welches mit unfokussierter Strahlung und synthetischer Apertur arbeitet, wird verwendet um die Vorteile von GPU-Algorithmen zur Rekonstruktion von großen THz Datensätzen zu zeigen. Weiterhin wird eine volumenbasierte Simulation für dieses System präsentiert. Zusätzlich wird eine geometrische Konfiguration und das Rendering von multimodalen Daten des THz Setups vorgestellt.

CONTENTS

1 INTRODUCTION 1
- 1.1 Motivation 1
- 1.2 Challenges 1
- 1.3 Contribution 2
- 1.4 Outline 2

2 FUNDAMENTALS 5
- 2.1 Computer Graphics 5
  - 2.1.1 3D Representations 5
  - 2.1.2 Rendering 9
  - 2.1.3 Illumination 13
  - 2.1.4 GPGPU 17
- 2.2 Terahertz Radiation 19
  - 2.2.1 Electromagnetic Spectrum 19
  - 2.2.2 Imaging Systems 20
  - 2.2.3 Applications 22
- 2.3 Requirements 22
  - 2.3.1 Simulating THz Radiation 22
  - 2.3.2 Processing THz Data 24

3 SPARSE VOXEL TREES 25
- 3.1 Motivation 25
- 3.2 Concept for THz Simulations 26
- 3.3 Data Structure 30
  - 3.3.1 Related Work 30
  - 3.3.2 Generalization of SVOs to SVTs 33
  - 3.3.3 Attribute Compression 36
  - 3.3.4 Memory Consumption 37
- 3.4 Voxelization 40
  - 3.4.1 Overview 40
  - 3.4.2 Related Work 40
  - 3.4.3 Algorithm Overview 42
  - 3.4.4 Triangle Stream Processing 43
  - 3.4.5 Voxel Stream Processing 49
  - 3.4.6 Results 53
- 3.5 Rendering 65
  - 3.5.1 Overview 65
  - 3.5.2 Related Work 65
  - 3.5.3 Algorithm Overview 67
  - 3.5.4 Ray Traversal 69
  - 3.5.5 Optimizations 77
  - 3.5.6 Results 86

3.6 Summary 95

4 SIMULATION OF WAVE EFFECTS 97
4.1 Motivation 97
4.2 Overview of Simulation Methods 98
4.3 Related Work 98
4.4 Main Influences on a THz Simulation 99
4.4.1 Wave Properties 99
4.4.2 Object Interaction 101
4.4.3 Acquisition System 103
4.5 THz Simulations 104
4.5.1 Hybrid Setup Simulation 105
4.5.2 THz Simulation with SVTs 115
4.6 Summary 133

5 DESCRIPTION OF THZ IMAGING SYSTEMS 135
5.1 Motivation 135
5.2 Hybrid 3D Scanning System 136
5.2.1 System 136
5.2.2 Configuration 137
5.2.3 Geometrical Calibration 139
5.2.4 Reconstruction on GPU 145
5.2.5 3D Image Generation 148
5.3 Pixelwise Scanning System 152
5.3.1 System 152
5.3.2 Configuration 154
5.4 Summary 155

6 CONCLUSION 157
6.1 Summary 157
6.2 Future Work 158
6.2.1 Adaptive and Palette-Based SVTs 158
6.2.2 Ray Queue for Multi-Bounce Simulations 159
6.2.3 SVTs without Triangles and Rays 159

BIBLIOGRAPHY 161

LIST OF FIGURES

Figure 2.1 Texture mappings for more surface details 6
Figure 2.2 A common example for octree usage 8
Figure 2.3 Octree variants 9
Figure 2.4 Coordinate transformations of a rasterization pipeline 10
Figure 2.5 Rasterization and fragment shading 11
Figure 2.6 Maximum intensity projection and isosurface rendering 12
Figure 2.7 Parameters of illumination models 14
Figure 2.8 Microfacets, shadowing and masking 15
Figure 2.9 The spectral band of THz radiation 19
Figure 3.1 Voxel rendering variants 28
Figure 3.2 Sampling estimation for varying N 29
Figure 3.3 SVO structure of [LK10] 31
Figure 3.4 Hierarchical voxel slabs of [LK10] 32
Figure 3.5 N^3-tree of [LHN05] 32
Figure 3.6 Node representation of [CNLE09] 33
Figure 3.7 Generalizing SVOs of [LK10] to SVTs 34
Figure 3.8 Generalizing the Morton order 35
Figure 3.9 Color compression 36
Figure 3.10 Normal compression of [CDE*14] 36
Figure 3.11 Memory structure of the SVT 38
Figure 3.12 N influences the memory consumption 39
Figure 3.13 Overview of the voxelization approach 44
Figure 3.14 Triangle subdivision cases 45
Figure 3.15 Functionality of the stitch queue buffer 52
Figure 3.16 Scenes for evaluation of the SVT creation 54
Figure 3.17 Comparing attributes of SVT creation with SVO approaches 59
Figure 3.18 Parameter influence on memory usage 60
Figure 3.19 Parameter influence on triangles per batch 60
Figure 3.20 Parameter influence on number of voxel-attribute pairs per batch 61
Figure 3.21 Parameter influence on timings 61
Figure 3.22 Tree traversal of [CNLE09] with kd-restart of [HSHH07] 67
Figure 3.23 Intersection of ray and brick with bitmasks 72
Figure 3.24 Voxel ray traversal of [AW87] 74
Figure 3.25 Push() and Pop() for stack-based traversal 76
Figure 3.26 Idea of beam optimization from [LK10] 78

Figure 3.27 Using dilation to compensate error of beam optimization 79
Figure 3.28 Stopping criteria of beam optimization and early exit 81
Figure 3.29 Influence of N to parent attributes 82
Figure 3.30 Level-of-detail popping of early exit 83
Figure 3.31 Example for the use of bitstacks 84
Figure 3.32 Used views of evaluation 86
Figure 3.33 Views of volume rendering evaluation 90
Figure 3.34 Influence of ray length on performance 93
Figure 4.1 Constructive and destructive interference 100
Figure 4.2 Polarization variants 100
Figure 4.3 Diffraction and Huygen-Fresnel Principle 101
Figure 4.4 Radiation pattern of an antenna 103
Figure 4.5 Beam parameter product, Rayleigh length 104
Figure 4.6 Parameters for the reflection mechanisms 107
Figure 4.7 Influence of σ_h and L_c 108
Figure 4.8 Parameters for focusing and spotlight 109
Figure 4.9 Influence of $\theta_{Tx/Rx}$ 110
Figure 4.10 Roughness influences reconstruction 112
Figure 4.11 Geometrical properties of a THz setup influence the imaging 112
Figure 4.12 Adjustable antenna properties 113
Figure 4.13 Comparing simulation and measurement 114
Figure 4.14 Thickness voxelization with triangles 116
Figure 4.15 Layer creation and error of postponed evaluation of physical behavior 117
Figure 4.16 Gaussian beam with ray bending 120
Figure 4.17 Iterating material layers 122
Figure 4.18 Phase maps for a varying number of rays 126
Figure 4.19 Rendered SVT scenes for evaluation 127
Figure 4.20 Influence of roughness in the *Rough* scene 130
Figure 4.21 The *Usaf* scene shows influences of the focusing and the radiation pattern 131
Figure 4.22 Frequency-dependent reflection behavior in the *Crystal* scene 132
Figure 4.23 Frequency-dependent transmittance behavior in the *Crystal* scene 132
Figure 4.24 Influence of r_{lens} in the *Crystal* scene 133
Figure 5.1 Prototype for hybrid 3D THz imaging 136
Figure 5.2 Configuring hybrid imaging setups 138
Figure 5.3 Measurement by laser distance meter 140
Figure 5.4 Pillar scene for geometrical calibration 141
Figure 5.5 Determination of angular slice with maximum intensity 142
Figure 5.6 Parameters for finding U_v by optimization 143

Figure 5.7 Calibration of THz and optical camera 144
Figure 5.8 Efficiency of precalculated distances 147
Figure 5.9 Alternative renderings of pure slices 149
Figure 5.10 Interpolation for fusing intensity values 150
Figure 5.11 Rendering of fused volume grids 151
Figure 5.12 Volume rendering with multimodal overlay 152
Figure 5.13 Pixelwise Scanning System 153

LIST OF TABLES

Table 3.1 Memory consumption for varying N 39
Table 3.2 Comparing performance of SVT creation with SVO approaches 55
Table 3.3 Memory consumption and performance breakdown for performance comparison 56
Table 3.4 Triangle counts for varying $K^{max}_{vox/tri}$ 62
Table 3.5 Influence of N on memory consumption 63
Table 3.6 Influence of implementation details on memory consumption 64
Table 3.7 Evaluating the influence of N on performance for SVT creation 65
Table 3.8 Variables for ray traversal of SVTs 71
Table 3.9 Comparing rendering performance of [LK10] and SVT 88
Table 3.10 Evaluating the influence of early exit and beam optimization 89
Table 3.11 Performance of SVTs for volume rendering 92
Table 3.12 Influence of ray iterations on performance 94
Table 4.1 Statistical surface roughness parameters 102
Table 4.2 Differences between [Kli12] and THz simulation with SVTs 124
Table 4.3 Setup properties for simulating THz radiation with SVT rendering 125
Table 4.4 Statistics of THz scenes 128
Table 4.5 Influence of material and thickness on the THz simulation with SVTs 129
Table 4.6 Signals of different roughness values 129
Table 5.1 Parameters of the system configuration 137

1 INTRODUCTION

1.1 MOTIVATION

Simulating light propagation and processing massive data in a performant manner are two main research areas in computer graphics (CG). Well established approaches for both disciplines and different applications exist (e.g. [Vito1, RDGK12]). Terahertz (THz) imaging as a very promising modality is an interesting application, because it provides new challenges for these research fields.

The simulation of THz radiation allows a prototyping and evaluation of imaging quality of systems which are not built physically. Therefore, production costs can be reduced and the imaging quality can be improved before an expensive system is produced. The efficient processing of massive THz data leads to performant imaging systems, which are needed for applications like security screenings or material testing in mass production.

1.2 CHALLENGES

Regarding performant illumination methods, valid approximations for visible light become incorrect for THz radiation. These approximations solely depend on ray optics, but they are only valid if the wavelengths of the radiation are much smaller than the radiated surface irregularities [ST07]. Wavelengths of THz radiation reach the dimensions of real-world surfaces and effects that are explained by wave optics can not be neglected because they influence the imaging. Therefore, physical effects like scattering from rough surfaces or interference need to be included for a more correct simulation of THz radiation. Additionally, a finer ray sampling and highly detailed surface representations are required to incorporate these effects correctly. The processing of massive data becomes important here. A performant creation and processing of scene representations with a high level-of-detail are necessary. Furthermore, THz imaging setups are not standardized yet (cf. [FvSB*11]). Individual system descriptions are used to reconstruct and fuse the measured sensor data. To allow a performant processing of massive data in these cases, an efficient geometrical configuration of the THz system is necessary.

1.3 CONTRIBUTION

The following contributions to the aforementioned challenges are presented in this thesis:

- Based on sparse voxel octrees (SVOs), a generalized sparse voxel tree (SVT) structure has been developed. The reduction of memory requirements for a surface representation allows a higher level-of-detail by increasing the resolution of the represented surface. The implemented rendering of these SVTs allows a faster processing and improves the performance of a simulation accordingly. For the special case of octrees, an out-of-core construction on a graphics processing unit (GPU) has been published in [PK15].

- By utilizing the processing speed of GPUs, fast simulation methods of several physical properties that influence THz imaging systems have been developed. Mainly, the roughness of irradiated surfaces and the properties of THz antennas have been considered. Parts of these simulation methods are published in [PKK*13].

- The performance of a two-dimensional synthetic reconstruction of massive THz data has been improved. It allows a near real-time processing by the reduction of computational overhead and the use of the GPU. The approach is published in [KKP*12] and [PKK*13].

- A geometrical configuration and calibration for THz scanning systems has been developed. For the use case of a prototypical scanner, it allows to fuse reconstructed surface profiles to a consistent 3D representation of the scanned object. Furthermore, a geometrical calibration for multimodal processing is used to render the reconstructed 3D volume with an overlay of a video image. Partially, the results have been published in [PKK*13].

1.4 OUTLINE

After this introduction, chapter 2 provides the fundamentals for the dissertation. Furthermore, requirements for the usage of graphical models and simulation for THz imaging are evaluated.

Chapter 3 introduces a representation of generalized high resolution sparse voxel trees. First, the data structure of these sparse voxel trees is discussed. Afterwards, a performant out-of-core creation on GPU and the rendering of these voxel trees is presented.

The topic of chapter 4 is the application of computer graphics methods to the simulation of electromagnetic radiation in respect to properties of THz radiation and the imaging system.

The system description of two individual THz imaging systems is given in chapter 5. Specially, the geometrical configuration of different systems is discussed. Additionally, a performant reconstruction of massive data on GPU is presented. The fusion of reconstructed datasets to a consistent 3D representation and a rendering of multimodal data is shown as well.

Chapter 6 summarizes the thesis and discusses possible ideas for future work.

2 FUNDAMENTALS

The following chapter provides necessary fundamentals for this thesis. An overview of relevant CG principles is given in Sec. 2.1. They include overviews of surface representations, rendering techniques, illumination models and GPU-processing. These topics serve as a foundation for the proposed approaches. It follows an introduction to THz radiation and THz imaging to present the field of application (see Sec. 2.2). An analysis of requirements for applying CG methods to THz applications is given in Sec. 2.3. While this chapter discusses basics for the specific aspects of this thesis, [SM09] and [Lee09] are recommended as introductory references to get more detailed information on general CG and THz methods, respectively.

2.1 COMPUTER GRAPHICS

2.1.1 *3D Representations*

3D objects are represented by triangles or volumes

The 3D representation of an object is a basic element of CG, because it is required as input, intermediate result or output for all CG approaches depending on the application. The two most common possibilities to describe 3D objects are triangle meshes and volume grids. While triangle meshes are used to represent the surface of objects, volumes are used to represent density information with a spatial extent. Brief overviews of triangle meshes and volume grids are provided in Sec. 2.1.1.1 and Sec. 2.1.1.2, respectively. Due to the proposed approach of this thesis, Sec. 2.1.1.3 discusses voxel tree structures which are a special form of volume grids.

2.1.1.1 *Triangle Meshes*

Triangle mesh approximates the surface

Triangle meshes are the most common data representation of 3D objects in CG, because most applications do not need information about the inner parts of objects. This representation saves memory and bandwidth by creating only hulls of the objects. Such a hull is created by a mesh which consists of a set of triangles. These triangles approximate the original surfaces piecewise. If more triangles are available, a more precise surface approximation can be achieved.

Each triangle has three vertices with additional attributes

Each triangle in Cartesian space is defined by three points or vertices which form a unique plane from three connected line segments if the vertices are not lying on the same line. While a triangle soup only holds this information of triangle positions, a triangle mesh stores additional information for the adjacency of triangles and al-

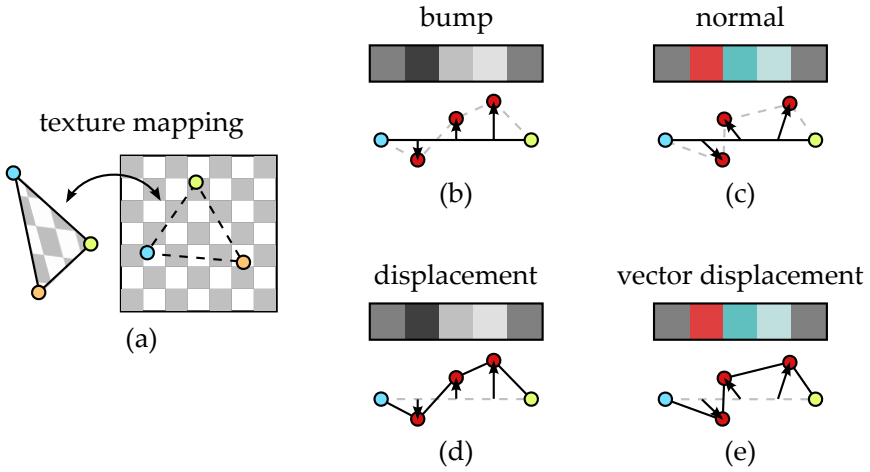


Figure 2.1: Vertex attributes which map to texture coordinates of an image improve the visual appearance of the triangle (a). Surface details can be increased, if the referred texture colors serve as a surface description. Bump mapping is using a grayscale image which is interpreted as heightmap. Inner points of the triangle are treated as if the are shifted along the triangle normal by the given height (b). Normal mapping uses RGB channels to simulate a moving of the points along an arbitrary direction (c). For common displacement mapping, vertices inside the triangle are really moved along the normal direction by heights to change the surface geometry (d). Vector displacement mapping extends displacement mapping and moves vertices along arbitrary directions (e).

lows to change surfaces by operations on the resulting set of connected triangles. For visualization and rendering purposes, the vertices of a triangle have further attributes in form of coordinates which map to other representations. The detail of the approximated surface is increased or the visual appearance is enhanced by keeping the triangle count constant.

Bump, normal and displacement mapping add surface details

Texture coordinates which map a color to each point on the triangle by an attached texture (see Fig. 2.1 (a)) or normals which allow an illumination calculation are the most common vertex attributes. In respect of a THz simulation, bump, normal and displacement mapping are mentionable (see Fig. 2.1 (b) - 2.1 (d) for a comparison), because these techniques allow to add random detail on surfaces which can be used for a more realistic scattering simulation. All three mappings have in common that a texture value represents surface properties. While a bump map interprets the values as heights in the illumination calculation, a normal map interprets these values as directions. In both cases, the appearance of the triangle looks more detailed in the rendering, but if the triangle is oriented parallel to the camera direction, the flat geometry becomes obvious. The most realistic behavior is provided by displacement maps, because the geometry is modified accordingly. Here, additional vertices in the triangle plane

are moved along the values in the texture. While common displacement maps are used to move the vertices along the surface normal of the triangle by a height map, vector displacement maps allow to move the vertices along an arbitrary vector direction. A true enhancement of surface details is the result when rendering the mesh with these displacement techniques.

Literature on geometry processing

A large number of further methods, techniques and algorithms related to triangle and mesh processing are available, but they are out of scope of this thesis. More detailed and comprehensive information on triangle meshes can be found in [Ede06] or [Pre93].

2.1.1.2 *Volume Grid*

Volumes contain scalars, vectors or tensors

Imaging modalities like x-ray or ultrasound allow to determine inner structures of 3D objects. Since triangle meshes cannot be used to represent such a dataset, because they only represent object hulls, a volume grid is used. It allows to store information of the inner part of the object. Usually, a Cartesian grid is used to represent a value in each of its cells. This value can represent different properties. For rendering of inner objects structures, this value is usually a one-dimensional density or intensity and the volume grid represents a 3D scalar field. For other applications, a volume grid can also be used to store vectors and tensors. Therefore, many rendering techniques for visualizing volumetric datasets exist (see Sec. 2.1.2.2).

Voxel definition and advantage of voxel trees

The smallest subspace inside the Cartesian volume grid is called a voxel. The possibility to represent inner structures or spatially varying properties requires a value at each voxel. Therefore, the memory requirement of regular volume grids is very high, so that object representations at high resolutions become impractical. Since a large number of voxels is empty and the grid has only a sparse distribution of relevant values in most real-world cases, the memory consumption can be decreased by a more efficient representation of sparsity. Voxel tree structures remove the need for storing large empty spaces, because they store only single tree nodes which represent larger blocks of empty space. This description of empty space reduces the memory consumption drastically.

2.1.1.3 *Voxel Tree Structures*

Octree definition

The most common representation of voxels in a tree structure is done by octrees. In general, an octree is a hierarchical data structure to allow a more efficient solving of problems by a divide and conquer strategy. Spatial datasets are divided into eight equally sized subsets, which represent new child nodes of the root node. Those nodes are recursively divided to eight or zero child nodes until the parts of the main problem are faster to process than the original problem.

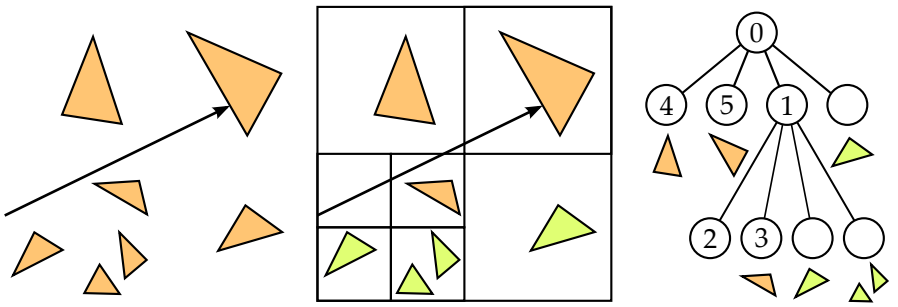


Figure 2.2: The acceleration of raytracing is a common application of octrees: With a naive approach all triangles need to be tested for intersection with one ray (a). By the use of an octree, only orange triangles need to be tested for intersection (b). The numbers show the order of visited octree nodes in the hierarchy and the corresponding triangles (c).

Octree usage for acceleration of raytracing

A common example is the use of an octree as data structure for improving the performance of rendering systems based on raytracing. With an naive approach, the intersections between rays and triangles need to be found by testing all rays against all triangles. With an octree, the full set of triangles is spatially sorted into subsets, but instead of testing all triangles for one ray, the ray traverses the octree structure and checks only those triangles that are related to the intersected octree nodes. As a result, the performance increases because the traversal of the octree nodes is more efficient than testing all triangles. Fig. 2.2 shows an example of an 2D octree, i.e. a quadtree. In comparison to all triangles in Fig. 2.2 (a), the green triangles in Fig. 2.2 (b) and (c) do not need to be tested for intersection.

Definition of voxel octree and sparse voxel octree (SVO)

Although the octree is usually used as an acceleration structure and its nodes link to subsets of the original data, it can be used to store the data directly as well. If each node stores only information about one element, the node can be interpreted as a voxel and the octree becomes a voxel octree (see Fig. 2.3 (a) and (b)). By removing the limitation of maintaining either all eight subnodes or no subnode, the sparsity of the scene can be exploited more efficiently, because only those subnodes which contain additional scene information are stored. Therefore, the voxel octree becomes a sparse voxel octree (SVO) and does not maintain empty child nodes of a parent anymore (see hierarchy in Fig. 2.3 (b) without dashed nodes).

Definition of an N^3-tree

A generalized form of the octree is an N^3-tree (see Fig. 2.3 (c) and (d)). The main difference between both structures lies in the number of children per node. While the number of child nodes is fixed to eight for the octree, it varies for the N^3-tree. Here, each node can have N subdivisions along one dimension, which results in N^3 child nodes for a 3D scene description. Those child nodes of one parent

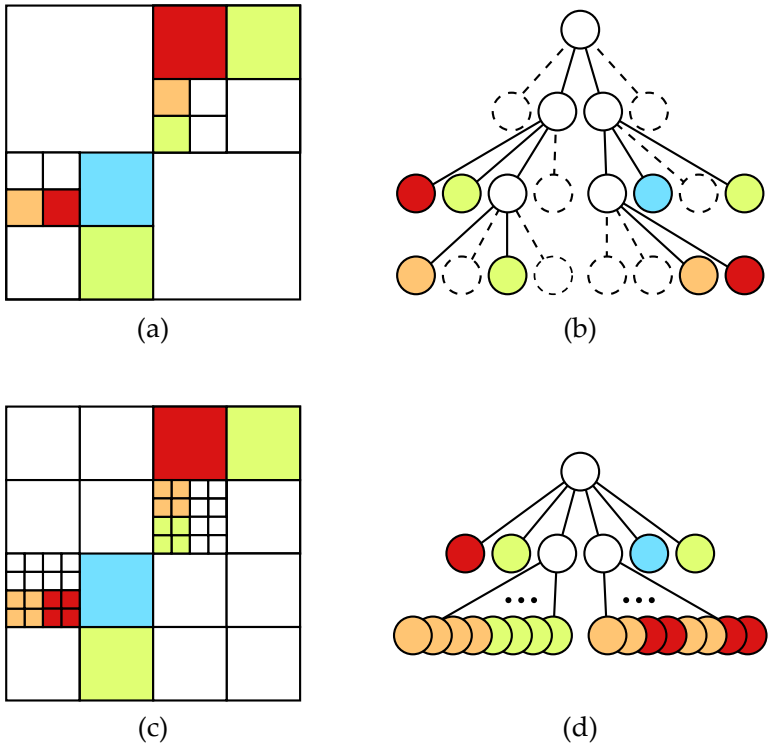


Figure 2.3: Different variants of octrees are shown: Interpreting nodes of an octree as single elements leads to a voxel octree (a). The corresponding hierarchy can be improved by removing the dashed nodes. Without these nodes, a sparse voxel octree is obtained (b). Increasing the number of child nodes in each dimension leads to a sparse N^3-tree. As an example, a 4^2-tree and its hierarchy are depicted (c,d).

are usually called bricks and are stored in a 3D-texture block, if appearance attributes are stored per child node.

2.1.2 *Rendering*

Rasterization or raycasting are most common

A visual representation of 3D objects and surface structures is obtained by rendering. The two most common techniques are the use of a rasterization pipeline and raycasting which are discussed in the following. The approaches of this thesis are based on raycasting, but a short discussion on the rasterization pipeline is provided as well, because the proposed voxel processing of Chap. 3 can be interpreted as a general pipeline for rasterization. Triangle meshes serve as input for a 3D rasterization to the voxel tree structure. Afterwards a ray-based rendering creates the visual representation on the screen.

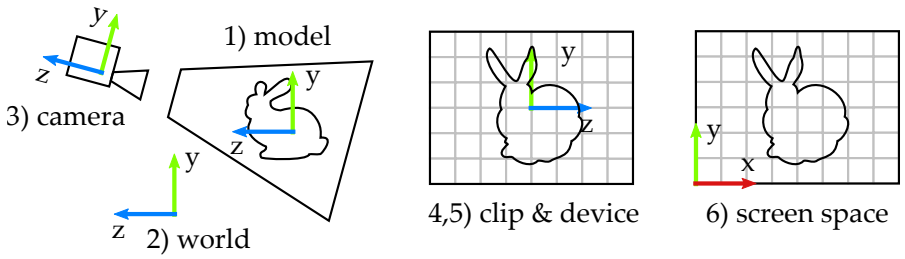


Figure 2.4: The coordinate transformations of the rasterization pipeline are depicted. The model or local coordinates of the geometries are transformed to world coordinates which lead to camera coordinates afterwards. A perspective transformation leads to clipping coordinates which are transformed to normalized device coordinates. A viewport transformation provides the final coordinates in screen space.

2.1.2.1 *Rasterization Pipeline*

Pipeline has a fixed sequence of operations

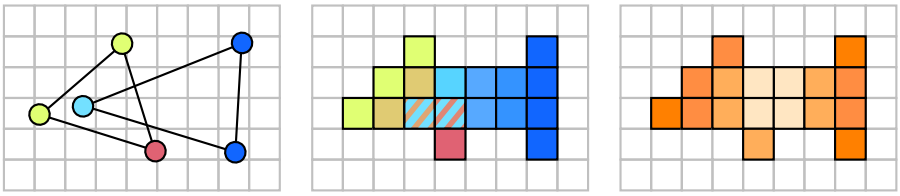
One possibility to render triangle meshes is the highly optimized rasterization pipeline approach of GPUs. Usually, the pipeline consists of fixed sequential steps which are required when creating a 2D visualization of a 3D representation. Although current GPUs and APIs allow a flexible usage of the pipeline by skipping or extending some of these steps, the basic operations of the pipeline concept do not change.

Coordinate transformations are applied to the 3D representation

First, model transformations are done which are depicted in Fig. 2.4. Each 3D object is defined in local coordinates and needs to be transformed to world coordinates, so that all objects have the same reference system. Next, a camera transformation is required to represent the 3D object in relation to the camera which shows the scene. After this transformation, the origin of the coordinate system lies in the center of the camera and the 3D scene is represented in camera coordinates. In the next step the projection transformation is applied. According to the desired camera properties a perspective view or an orthogonal view of the scene are usually used. The scene is represented in clip coordinates afterwards. The outer parts of the triangles are clipped at this stage so that only relevant triangle parts are left for rendering. With a following perspective division, the clip coordinates are transformed into normalized device coordinates in the range $[-1, 1]$ for x-, y- an z-axis. Afterwards the viewport transformation finalizes the transformation to screen space coordinates so that x- and y-axis represent width and height of the view in pixels, respectively. The z-axis is in the range of [0,1] which allows a depth sorting to calculate hidden surfaces.

Rasterization determines fragments for a pixel

In the rasterization stage after the coordinate transformations, it is determined which pixels on the screen will be covered by the individual triangles (see Fig. 2.5 (a,b)). Furthermore, the vertex attributes



(a) (b) (c)

Figure 2.5: After the final transformation to screen space (a), the rasterization is determining fragments and interpolating vertex attributes of the triangles (b). In the final step, the final pixel color is determined from the generated fragments per pixel (c).

of the triangle are interpolated to determine the contribution of a triangle attribute to the pixel. This contribution per pixel is called a fragment, because triangles can overlap which leads to a set of contributions per pixel.

Fragment shading calculates the final pixel color

The final determination of pixel colors is done by fragment shading or fragment processing (see Fig. 2.5 (c)). Here, fragments of a pixel are combined by specific operations. The most common operation is the depth test to show the fragment value of the triangle which is nearest to the camera. By rasterizing triangles, a depth value for each resulting pixel is calculated. If the generated depth value is behind the stored depth value in the depth buffer, the fragment is discarded. If the depth value is in front, the depth buffer and the pixel color are updated with the values of the current triangle. Other common applications are anti-aliasing, illumination, deferred shading, blending or stencil tests, but since the fragment shader is fully programmable many other algorithms exist.

2.1.2.2 *Raycasting*

Volumes and triangles can be rendered by traversing rays

In comparison to the rasterization pipeline, raycasting or raytracing is more flexible because the computation of the ray traversal can be adapted to specific needs of an image generation. The main idea of raycasting is the calculation of intersections with the given scene representation by a ray traversal, i.e. a virtual camera is sampling a scene by a discrete number of rays which reconstruct the appearance of the 3D object (cf. Fig. 2.2 (a)). In the case of triangles as input, raycasting is less common, because current GPUs are designed and highly optimized for an optimal triangle rendering based on the pipeline approach.

Raycasting is the main technique for volume rendering

In case of volume rendering, raycasting provides a very generic functionality for visualization. Here, a ray samples the volume uniformly to obtain a number of discrete values which are calculated by trilinear interpolation. Usually, each interpolated sample is mapped to a color value and an opacity value by a transfer function. During

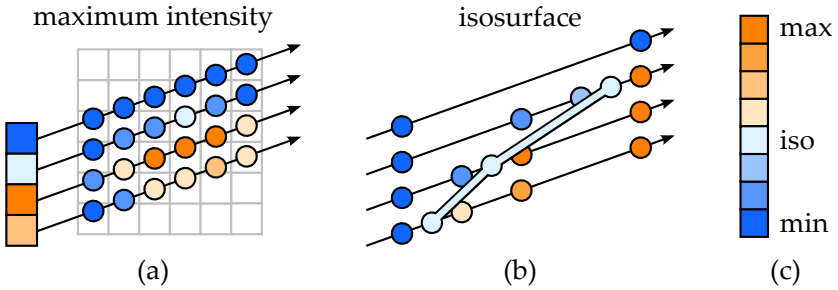


Figure 2.6: The principles of maximum intensity projection and isosurface rendering are depicted. The final pixel color is set to the maximum value along the ray which is determined by comparing current sample and current maximum value of all previous samples. If the current sample is larger, it becomes the new maximum of the ray (a). To obtain an isosurface rendering, samples along the ray are compared with the isovalue. If the isovalue lies between two neighboring samples, the ray segment is recursively sampled in the center of the ray segment which holds the isovalue. If a defined precision is achieved, the interval bisection is terminated (b). A exemplary mapping of values is used to illustrate both raycasting approaches (c).

the ray traversal, the sampled values are integrated from back to front and composited to a final output color. In addition, physical models for the simulation of light distribution are used during this integration for more realistic or meaningful representations. This general volume processing is adjusted and extended by many volume visualization techniques and principles. An in-depth introduction to these methods of volume rendering can be found in [EHK*06].

Maximum intensity projection

Maximum intensity projection and isosurface rendering are two volume rendering techniques which are discussed in the following, because they are applied in this thesis. Fig. 2.6 shows both rendering approaches. For maximum intensity projection, each sample on the ray is compared with the maximum value of all samples that have been encountered by the ray already. If the current sample is larger, it gets the new maximum. After the ray traversed the volume, it holds the maximum value of all samples which is mapped to a color. If the sampled volume grid contains intensities, the color representation allows to identify regions which are relevant in the respective application. In the context of THz security screenings, it allows to visualize hidden metallic objects like weapons, because they have a high reflectivity in the THz range (see Sec. 5.2.5.3).

Isosurface rendering

For isosurface rendering, the samples along a ray are compared to an isovalue. This isothreshold serves as the definition of the surface which should be identified in the volume. To obtain this representation, the current and the previous sample of the ray are interpreted as a range. If the isovalue lies in this range, the surface is found. To get

a more precise determination of the surface and to reduce visual artifacts an interval bisection is applied. Here, the range is split into two new ranges by sampling the ray in the middle between the original samples. Both new ranges are checked again whether they contain the isovalue. The range that holds the isothreshold is checked further by splitting the range again. This process is repeated until the desired precision for visualizing the surface is achieved.

2.1.3 *Illumination*

Physically plausible results are obtained by raytracing

The visualization of a CG scene is mainly influenced by the used simulation of visible light. Usually, an illumination model in the rendering process is used to calculate pixel colors. Many illumination techniques evolved during the last centuries to improve the physical plausibility and the performance of the simulation. Raytracing is one of the fundamental methods for most of these techniques, because it allows to approximate visible light by infinitesimal rays. Due to the small wavelengths of visible light in comparison to surface properties of real-world scenes, the rays serve as the basic element for calculating the illumination by geometrical optics.

Difference between local and global illumination

CG simulations of visible light can be categorized into local and global illumination models. Local illumination models focus on the fast calculation of the most important influences on the visual appearance, i.e. the simulation of primary reflections with the most important light contribution and shadows are the main subject of these models. Global illumination models focus on the physical plausibility but perform slower in comparison to local illumination methods. In addition to primary reflections and shadows, secondary reflections are calculated. Therefore, physical effects like caustics, diffuse reflections or sub-surface scattering can be simulated. Current research in this field even extends these methods by taking effects of wave optics into account (e.g. [CHB*12, SML*12]).

Further readings

For a basic understanding of CG illumination models and the proposed adaptions in Chap. 4, a subset of the most relevant principles is discussed in the following. Further introductions can be found in [SM09] and [AMHH08]. More comprehensive information on the topic of physically based rendering can be found in [PHJ16] and [DBB06].

2.1.3.1 *Local Illumination*

Phong model

The Phong model ([Pho75]) is one of the standard models for calculating local illumination. The intensity of the light at each point on a

$\vec{n}$: surface normal
$\vec{h}$: halfway
$\vec{v}$: view
$\vec{l}$: light
$\vec{r}$: reflection
p : point on surface
θ : $\arccos(\vec{n} \cdot \vec{l})$
α : $\arccos(\vec{v} \cdot \vec{r})$
β : $\arccos(\vec{n} \cdot \vec{h})$

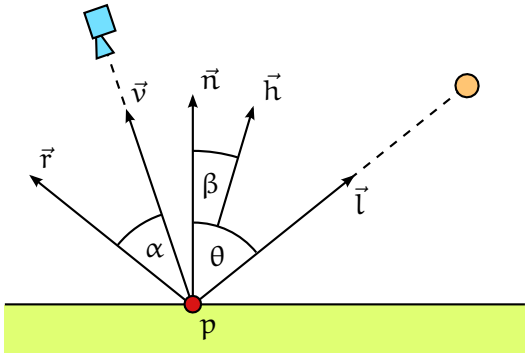


Figure 2.7: The geometrical parameters of the discussed illumination models are shown. All vectors are normalized.

surface I_p is empirically defined by a sum of an ambient term I_a, a diffuse term I_d and a specular term I_s (see Eq. 2.1).

$$I_p = I_a + \sum_{m=1}^{\#lights} (I_d(m) + I_s(m)) \tag{2.1}$$

The final intensity has an ambient, a diffuse and a specular part

I_a serves as a constant factor for simulating a global light component for the whole scene. I_d and I_s are influenced by the spatial positioning of scene elements, surface orientations, light properties and a simplified material description to calculate an individual light contribution for each point in the scene. I_d represents the diffuse part of the light intensity which is independent of the viewing position, while I_s represents the specular part which depends on the viewing position. The calculation of I_d and I_s are shown in Eq. 2.2 and Eq. 2.3, respectively. Fig. 2.7 illustrates the geometrical parameters.

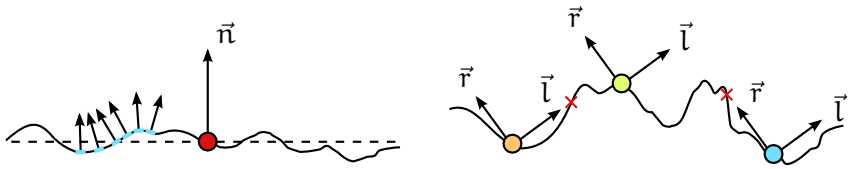
$$I_d(m) = k_d \cdot i_d(m) \cdot \cos(\theta(m)) \tag{2.2}$$
$$I_s(m) = k_s \cdot i_s(m) \cdot \cos^n(\alpha(m)) \tag{2.3}$$

Diffuse reflection

The intensity of the diffuse reflection I_d is calculated by Lambertian reflection. Therefore, the diffuse part of the intensity of the light source $i_d(m)$ is multiplied by the ratio of reflected diffuse radiation from the material k_d. This result is scaled by the dot product of normalized light direction $\vec{l}(m)$ and surface normal $\vec{n}$ which corresponds to $\cos(\theta(m))$ where $\theta(m)$ is the angle between both directions.

Specular reflection

In comparison to I_d, the intensity of the specular reflection I_s is obtained by the specular part of the intensity of the light source $i_s(m)$ and the reflected specular radiation from the material k_s. This ratio of the specular radiation is multiplied by $\cos^n(\alpha(m))$ which is taking the viewing position and the behavior of glossy materials into account. $\alpha(m)$ is the angle between the normalized viewing direction $\vec{v}$ and the normalized direction of a perfect specular reflection $\vec{r}$. The exponent n allows to simulate the contribution of glossy reflections



(a) (b)

Figure 2.8: With the facet slope distribution D, a rough surface is approximated by microfacets which have individual normals (marked in blue), while the surface is flat in the macroscopic view (dashed line). An increasing s in Eq. 2.6 simulates steeper slopes with more varying normals of the microfacets so that a rougher appearance of the surface is achieved (a). The geometrical attenuation G is used to simulate self-shadowing effects of rough surfaces. Either the surface point does not receive the incoming light (shadowing: orange point) or the outgoing light is not reflected (masking: blue point). The green point receives and reflects light (b).

by varying the light distribution which is centered around the perfect specular reflection. An increasing n leads to a decreasing size of the specular highlight on the surface.

Cook-Torrance improves physical plausibility of specularity

The Cook-Torrance model is another important approach in the context of this thesis, because it incorporates necessary effects which influence a CG simulation of THz radiation as well. The physical plausibility of the specular component I_s is improved by considering Fresnel reflection F, a facet slope distribution D and a geometrical attenuation factor G. Eq. 2.4 shows the calculation of I_s.

$$I_s = \frac{F \cdot D \cdot G}{\pi \cdot (\vec{n} \cdot \vec{l}) \cdot (\vec{n} \cdot \vec{v})} \tag{2.4}$$

Fresnel term F

The Fresnel term F describes the refraction and reflection of electromagnetic radiation between different media at a perfectly smooth surface. Next to the angle of incident radiation, refractive indices η and extinction coefficients κ of the involved media are used to determine the behavior at the boundary. While it is possible to consider polarization and differ between intensity and field equations, simplified variants or approximations ([Sch94]) are used for CG simulations of visible light. For unpolarized light and $\kappa = 0$, [CT82] uses the calculation of Eq. 2.5.

$$F = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left(1 + \frac{(c(g+c)-1)^2}{(c(g-c)+1)^2}\right) \tag{2.5}$$
$$\text{with} \quad g = \sqrt{\eta^2 + c^2 - 1} \quad \text{and} \quad c = \vec{v} \cdot \vec{h}$$

Facet slope distribution D

The Cook-Torrance model is based on the assumption that a rough surface consists of several flat facets which are individually oriented (see Fig. 2.8 (a)). D represents a statistical facet slope distribution

to simulate the orientation of those individual facets. Therefore, the scattering behavior of rough surfaces is incorporated by D. The Beckmann distribution is used for D, because it covers many materials and even very rough surfaces. Depending on the root mean square slope of the surface s and the angle β between surface normal $\vec{n}$ and half angle vector $\vec{h}$ (see Fig. 2.7), D is given by Eq. 2.6. In [CT82] it is stated that the influence of the wavelength on the roughness scattering is omitted in D due to simplicity.

$$D = \frac{e^{-(\tan(\beta)/s)^2}}{s^2 \cdot \cos^4(\beta)} \quad \text{with } \beta = \arccos(\vec{n} \cdot \vec{h}) \tag{2.6}$$

Geometrical attenuation G

Under the assumption that the surface consists of many individual facets, self-shadowing effects may occur, i.e. only a fraction of the incoming and a fraction of the outgoing radiation can be used (see Fig. 2.8 (b)). To compensate these effects of shadowing and masking, G uses $\vec{n}, \vec{l}, \vec{v}$ and $\vec{h}$ to attenuate the reflected radiation by the purely geometrical calculation in Eq. 2.7.

$$G = \min\left(1, \frac{2(\vec{n} \cdot \vec{h})(\vec{n} \cdot \vec{v})}{(\vec{v} \cdot \vec{h})}, \frac{2(\vec{n} \cdot \vec{h})(\vec{n} \cdot \vec{l})}{(\vec{v} \cdot \vec{h})}\right) \tag{2.7}$$

Normalization factor and reflectance correction

π is used to correct bidirectional reflectance values for rough surfaces, because smooth material samples serve as a basis for measured reflectance values usually (cf. [CT82]). The dot products in the denominator are used for normalizing the reflectance by the orientation of the surface to the viewer and to the light. More detailed information on the described parameters of the Cook-Torrance model is provided in [CT82].

2.1.3.2 *Global Illumination*

Rendering equation of [Kaj86] is the basis for global illumination

The standard rendering equation for calculating global illumination was presented by [Kaj86]. It is commonly formulated as in Eq. 2.8 and Eq. 2.9. The influences of time, wavelength or polarization are omitted, but they can be integrated depending on the desired correctness of the simulation. In the equation is stated that the outgoing radiance L_o from a point p on the surface in the direction ω_o is consisting of the emitted radiance L_e and the reflected radiance L_r (see Eq. 2.8).

$$L_o(p, \vec{\omega}_o) = L_e(p, \vec{\omega}_o) + L_r(p, \vec{\omega}_o) \tag{2.8}$$

$$L_r(p, \vec{\omega}_o) = \int_\Omega f_r(\vec{\omega}_i, \vec{\omega}_o) \cdot L_i(p, \vec{\omega}_i) \cdot \cos(\theta_i) d\vec{\omega}_i \tag{2.9}$$

L_r is calculated by the incident light and a BRDF

The calculation of L_r is created by the irradiance at p and consists of the bidirectional reflectance distribution function (BRDF) f_r, the incoming radiation L_i and a geometrical term $\cos(\theta_i)$ (see Eq. 2.9). All incoming light contributions are considered by integrating over

the hemisphere Ω of the surface. L_i is multiplied by $\cos(\theta_i)$ to compute the irradiance to the surface. While L_i and $\cos(\theta_i)$ represent the incoming light contribution before the interaction with the surface, the BRDF describes the scattering behavior of the material, i.e. the relation of the outgoing radiance to the incoming irradiance.

Requirements for a physically plausible BRDF

Although the BRDF can be an arbitrary function, it needs to fulfill the following three requirements for a physically plausible description of the material reflectance:

- positivity:
  $f_r(\vec{\omega}_i, \vec{\omega}_o) \geqslant 0$
- symmetry (Helmholtz reciprocity):
  $f_r(\vec{\omega}_i, \vec{\omega}_o) = f_r(\vec{\omega}_o, \vec{\omega}_i)$
- conservation of energy:
  $\forall \omega_i : \int_\Omega f_r(\vec{\omega}_i, \vec{\omega}_o)\cos(\theta_o)d\omega_o \leqslant 1$

More complex BRDFs allow to simulate more physical effects

In addition to these requirements, the physical plausibility of a BRDF can be increased by considering further parameters. In the simplest form, it requires only $\vec{\omega}_i$ and $\vec{\omega}_o$. More complex BRDFs include a dependence of p for spatially varying reflectance properties on the surface or an additional point p' to incorporate a subsurface scattering. Furthermore, a wavelength λ can be considered for diffraction effects.

[Kaj86] proposes path tracing to solve the rendering equation

To solve the rendering equation, [Kaj86] proposes path tracing which combines a Monte Carlo integration with a ray-based approach. Each ray represents a sample for a numerical integration of the rendering equation. Depending on the desired quality of the rendering, a varying number of rays is sent through each pixel. Each ray forms a path by reflecting at intersected surfaces. A stochastic scattering approach is used to determine the new direction of the ray and which type of reflection is happening. By tracing the individual path through the scene, the calculated light distributions are accumulated. Furthermore, the visibility to all light sources is evaluated at each intersection. After a stopping criteria, e.g. a specific number of ray segments per path, is fulfilled, the contributions of the rays per pixel are averaged and lead to the final pixel color.

2.1.4 *GPGPU*

THz applications can be accelerated by GPGPU

The acceleration of algorithms by a GPU is frequently applied in CG especially in the cases of processing of massive data and heavy computations. Similarly, this applies to the most THz applications with independent acquisitions and massive measurement data, so that the GPU can be exploited to accelerate these applications. While this use of general-purpose computing on GPUs (GPGPU) has still some

drawbacks or limitations in comparison to CPU programming, a large performance gain can be expected if an algorithm provides the possibility to parallelize subtasks with individual threads, because the hardware design of a GPU is optimized for the main task of parallel vertex and pixel processing.

CUDA is used for all proposed methods of this thesis

All proposed methods of this thesis are implemented by CUDA which is an API for GPUs of Nvidia. It allows to use subsets of the programming languages C, C++ and Fortran which are extended by additional functionalities for parallel code execution on GPU. The code of the proposed methods is implemented by the C/C++ interface. An extensive introduction to CUDA programming is provided in [SK10].

Main influences on a performant GPGPU implementation

In the following, a rough idea on the complexity of implementing a performant GPGPU algorithm should be given. More in-depth information can be found in [KH12]. Three main factors for exploiting the high performance of GPUs are:

- Memory accesses
- Thread divergence
- Occupancy

Memory accesses

The CUDA API provides several memory regions on the GPU for processing data with varying latencies, scopes and sizes. The fastest data access is possible by *registers*, but the number of available registers is most limited. *Shared memory* is slightly slower but has a larger size and can be shared between threads in a subgroup (thread blocks) to exploit additional caching effects. *Constant memory* is available to all threads with a fast access through caching, but is read-only. *Global memory* provides the largest size and is available to all threads as well, but it has the worst latency. *Local memory* is accessible by only one thread and maps internally to global memory if the registers of a thread are not enough. It follows that an optimal use of the available memory regions is required for the best performance. While all data could be stored in global memory for simplicity, a separation of the data to the specific regions can lead to a better performance.

Thread divergence

A principle of GPU parallelism is the lockstep execution of threads in the smallest hardware unit which is called a *warp* in CUDA terms. In current GPUs, each warp can process up to 32 threads. If each thread has the same sequence of operations, all 32 threads can be processed in one run, but if those 32 threads have diverging tasks, a branching requires additional execution of operations. Therefore, threads in one warp wait for other threads and the performance drops.

Occupancy

Depending on the memory requirements of the thread operations and the memory restrictions of the hardware, the number of parallelly executable threads can be determined. The ratio between these

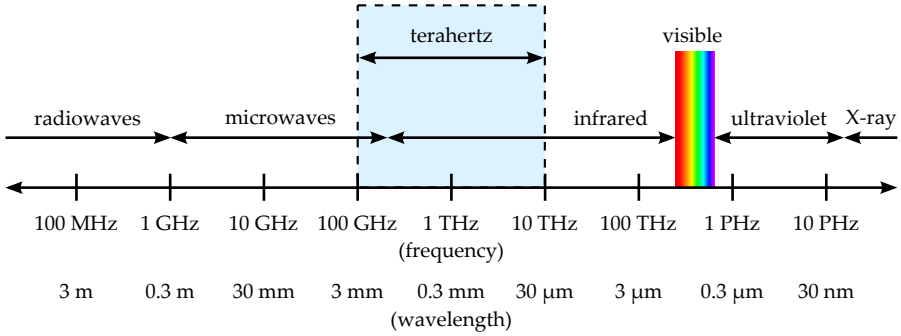


Figure 2.9: Parts of the electromagnetic spectrum and its bands are shown. The spectral band of terahertz radiation lies between microwaves and infrared.

threads and the maximum number of processable threads defined by hardware limitations can be interpreted as the occupancy of the GPU. It is a measure for the efficient exploitation of the GPU hardware. Due to other influences like memory latencies or the actual workload per thread, the performance is not necessarily increasing if the occupancy is increasing. A low occupancy only indicates a possible need to increase the number of parallelly executable threads by improving the operations inside a thread.

2.2 TERAHERTZ RADIATION

2.2.1 *Electromagnetic Spectrum*

Definition of THz radiation

Terahertz radiation is usually defined as the frequency range of the electromagnetic spectrum between 0.1 THz and 10 THz [AZ07, Lee09]. These frequencies are at the lower bound of infrared radiation and at the upper bound of microwave radiation (see Fig. 2.9). More precisely, they cover the bands of millimeter waves (0.03 - 0.3 THz, 1 - 10 mm), sub-millimeter waves (0.3 - 3 THz, 0.1 - 1 mm) and parts of the far infrared radiation (≈ 0.86 - 12 THz, ≈ 25 - 350 µm) [Lee09]. The imaging with higher frequencies is based on optical technology, while the imaging with lower frequencies is done with electronic technology. Just a few decades ago, both technological fields lacked of efficient detectors and sources for imaging in the spectral band of THz radiation. There are two main reasons for this so-called "THz gap" between the transition of electronics and optics. On the electronic side, suitable circuits that handle signals at high frequencies could not be built, while on the optical side, systems with a size close to the used wavelength could not be designed [Cha04]. In the last decades, more efficient technologies from both sides tried to close this THz gap. [Sie02] and [Ton07] provide overviews of these THz technologies.

Generation and detection of THz radiation

The generation of THz radiation can be done with optical or electronic technology. In comparison to systems on the electronic side, optical based measurement setups provide a more sophisticated material detection in the THz range due to higher bandwidths, but the size, complexity, and performance of these setups avoid the broad use in real-world applications currently. Electronic based systems are more flexible, because they are cheaper and smaller. They allow to acquire larger scenes and obtain additional spatial information of the scene in a more efficient way. Therefore, the proposed simulation methods of this thesis are more beneficial for prototyping of real-world systems that are based on electronic generation and detection of THz radiation. To process THz radiation with electronic technology, frequency multiplication of microwaves and heterodyne detection are usually used [Lee09]. By using diodes, different frequencies are mixed to generate or detect THz radiation with up- or down-conversion, respectively.

2.2.2 *Imaging Systems*

Active vs passive

A large variety of THz imaging systems were created in the last decades, because the design and setup of the system strongly depend on the application and the available THz technology. In general, THz imaging systems can be categorized into active and passive systems. While active systems use THz sources to irradiate the scene, passive systems detect natural atmospheric radiation only.

Coherency

Another property of the imaging system refers to the output. If the used radiation is incoherent, phase differences and frequencies vary and only intensity values can be obtained as output. If the radiation is coherent, phase differences and frequencies are constant. Here, phase and amplitude of the scattered electric field can be reconstructed.

Advantages of coherent radiation

The use of coherent radiation is beneficial because it allows to gain additional or better information by more sophisticated reconstruction methods. For example, depth information of the objects can be calculated in the signal processing or spectral material properties can be determined. Since atmospheric radiation in passive systems allows incoherent output only, the simulation concepts in this thesis focus on active imaging systems with coherent THz sources.

Radiation modes

Additionally, the imaging system can be characterized by the radiation mode, i.e. if pulsed or continuous wave (CW) technologies are used [Lee09]. While pulsed technologies transmit single pulses, CW technologies constantly radiate the scene. For the intended simulation of THz imaging systems with electronic technology (see Sec. 2.2.1), CW is advantageous because pulsed based technology reaches its limits in the frequencies of the upper spectral band of microwaves due to power requirements for very short pulses, which are necessary for obtaining scene information in a higher depth reso-

lution. Furthermore, the physical sampling of smaller pulses reaches current hardware limitations.

Advantages of FMCW

Specially, frequency modulated CW (FMCW) technology as part of CW technologies is beneficial for the proposed methods of this thesis, because it allows to reconstruct depth information which can be used for reconstructions of three-dimensional scene representations. Commonly, a frequency chirp is applied, i.e. several acquisitions for a sequence of frequencies in a frequency band are performed. An example for an FMCW setup which is combined with synthetic aperture imaging is discussed in [DKLB13].

Spatial positioning of THz sources and detectors

The positioning of sources and detectors strongly influences the imaging. If the screened object is between sources and detectors, the imaging is done by analyzing the transmitted radiation. If sources and detectors are on the same side of the object, the reflected radiation is analyzed. Furthermore, the number and relative arrangement of sources and detectors lead to different imaging possibilities. The simplest system in this context is the one with only one transceiver, i.e. one source and one detector are combined in one component. Here, a one-dimensional information is obtained. It contains the spectral response of the radiated area in the direction of the source radiation. If one transmitter and one receiver are used separately, the detection of reflected radiation is not limited to the source direction and the detection of transmitted radiation becomes possible. To obtain spectral information in spatially varying dimensions, two methods can be applied. The first method is scanning, i.e. a mechanical movement of the THz components, and the second method is the use of several transmitters and receivers, which is mostly done in synthetic imaging approaches.

Influences of positioning

The mechanical movement allows to place source and detector at varying positions. Each combination of the positions gives one measurement. Depending on the complexity of the mechanical scanning, these measurements can be fused to create one-, two- or even three-dimensional scene representations. An advantage of this method is the cost-efficiency, because current THz technology is more expensive than the equipment for a mechanical movement of components. The drawback is the performance, because the measurements are done sequentially. If the number of sources and detectors is increased for a system with focused acquisition and if the resulting signals do not influence each other, the spatially varying measurements can be done in parallel. Therefore, the performance of the system would increase, but the cost of the THz technology would be much higher. Although the performance increases, the data throughput would become a limiting factor for too many sources and detectors [KLD*10]. Therefore, recent hybrid scanning system designs try to combine a reduction of THz components with a high performance. The various spatial dimensions of the scene representation are reconstructed

differently. For example, one dimension is obtained by mechanical scanning, while another one is obtained by synthetic aperture imaging. Overviews of these systems can be found in [KLD*10]. A more focused overview of systems that allow real-time processing is presented in [FvSB*11].

2.2.3 *Applications*

Possibilities of THz detection

Although research of THz technology is still confronted with many challenges like atmospheric attenuation or more performant generation of terahertz radiation [Arm12], several applications for THz imaging exist and further promising applications are under development. One of the main goals is the detection of materials in several application areas. For example, in medicine it is used to detect illnesses like skin cancer by analyzing the absorption coefficient of the tissue (see [YFSPM12]). Another example is the detection of concealed objects for security applications. Metallic weapons can easily be seen, because metal has a high reflectivity in the THz range. THz spectroscopy is used for detecting other kinds of weapons, drugs or explosives. It allows to determine the spectral features of the imaged materials. These features need to be analyzed for a detection of specific materials then (see [FSH*05]). An overview of further applications like earth sensing or material inspection and quality control can be found in [DMBM05].

2.3 REQUIREMENTS

Motivation of using CG methods for THz imaging

Applying CG methods for THz imaging leads to benefits in both research disciplines. On the THz side, prototyping with a THz simulation framework is cheaper than an evaluation with a physically built system and computationally intensive parts of a processing framework benefit from improvements in performance by using GPU methods. On the CG side, a more general simulation of electromagnetic radiation broadens the scope of current illumination techniques and general CG methods for processing massive data can be improved in terms of performance and precision. Furthermore, the resulting methods can be used for other applications where similar problems need to be solved.

CG methods can be applied to a simulation of THz radiation or to a processing of measured THz data. These two applications have different requirements which are discussed in the following subsections.

2.3.1 *Simulating THz Radiation*

CG techniques gain performance due to approximations

CG techniques for physical simulations allow to generate correct or approximately correct results in a very performant manner. Fast il-

lumination approaches like final gathering or photon mapping are common approximations to simulate incoherent imaging with a propagation of electromagnetic radiation, but mainly physical effects that can be explained by ray optics are considered. For visible light, these approximations are sufficient to achieve a visually correct result, but THz imaging is additionally influenced by other physical effects that can only be described by wave optics.

Ray-based methods are partly invalid for THz

Therefore, common raytracing approaches provide less correct results if they are applied to a THz simulation. The reason for the strong influence of wave properties lies in the different regimes of the interaction between radiation and real-world surfaces. Wavelengths of visible light lie in the range of nanometers and are orders of magnitude smaller if they are compared to real-world surfaces in the millimeter range, while wavelengths of THz radiation are in the submillimeter range with similar sizes.

To simulate THz imaging with CG methods, it is required to incorporate wave properties of the electromagnetic radiation. Although ray-based approaches do not allow to cover all effects by default, they can be adapted to address the main features of wave optics. This adaption consists of two parts:

Efficient data structure for storing fine details and sparse volumes

- A new representation of scene objects is needed, because current surface representations (cf. Sec. 2.1.1) are not designed to cover the specific requirements of a THz imaging simulation. Roughness scattering requires highly detailed surfaces for a more correct calculation of the reflection, if wavelengths and surface details have similar dimensions. These surfaces can only be represented by a more memory efficient data structure. Mesh representations are memory efficient and can be used for highly detailed scenes, but they do not allow to store volumes of objects which would be necessary, because THz radiation penetrates different real-world materials. In this regard, meshes can only be extended by texture information that influence the raytracing because no information of the inner structures of the objects are stored. Volume representations can be used for simulating thickness by defining materials for each voxel, but they can not be used at a high resolution due to storage requirements of full grids. Furthermore, an efficient construction and rendering of such a scene representation is needed to guarantee a performant processing. To fulfill these requirements, a generalized sparse voxel representation is proposed in Sec. 3.

Simulating wave effects by adapting ray-based approaches

- To improve the correctness of THz simulations, existing raytracing approaches need to be extended by simulations of wave effects. The information per ray gets larger and more complex calculations need to be done. The additional effort leads to a performance decrease. Additionally, the implementation of a

coherent imaging simulation (see Sec. 2.2.2) decreases the performance as well, because the calculation of a scattered electric field with phase and amplitude information is necessary, while a calculation of intensities is sufficient for the simulation of incoherent imaging. To compensate the performance loss, the interaction of waves needs to be approximated. The physical wave properties and the correspondingly implemented approximations are discussed in Sec. 4.

2.3.2 *Processing THz Data*

Exploiting the GPU to allow a performant THz processing

A common CG method to improve the performance of an existing approach is the use of a GPU. Usually, a sequential processing is changed so that the processing can be executed in parallel. Depending on the possibility to parallelize the processing, the performance gain varies. Therefore, it is necessary to identify parallelizable processing steps and redundant calculations.

Geometrical configuration and calibration

For applying this principle to the processing of THz imaging data, it is necessary to geometrically describe and calibrate the imaging system. Due to the prototypical character of the setups, individual configurations need to be found. The configurations allow to define the dimensions that can be imaged in parallel. Depending on the properties and complexity of the measurement setup, this configuration further allows to create a more sophisticated scene representation in a more performant manner. The system description of exemplary THz imaging systems and the application of CG methods to individual processing steps are described in Sec. 5.

3 SPARSE VOXEL TREES

In this chapter, properties and processing of sparse voxel trees (SVTs) are discussed. First, a motivation for the use of voxels is given in Sec. 3.1. The concept of the SVT and its application for a THz simulation is discussed in Sec. 3.2. Sec. 3.3 addresses the data structure of SVTs and compares it with similar tree structures. A fast and efficient method for generating SVTs from massive triangle meshes by an out-of-core approach on GPU is presented in Sec. 3.4. While general rendering and processing of the resulting SVT structure are topics of Sec. 3.5, concrete adjustments for using SVTs in a THz simulation are discussed in Sec. 4.5.2.1 and Sec. 4.5.2.2. The chapter is summarized in Sec. 3.6.

3.1 MOTIVATION

Surface and volume rendering need to be combined

Requirements for simulating THz radiation with CG rendering techniques differ from usual requirements of surface rendering and volume rendering. If the interaction between THz radiation and real-world materials like fabric or tissue needs to be simulated, the radiation will penetrate surfaces and a common surface rendering approach would not be sufficient, because no information of the inner object structure is given. Hence, a more complex description of surface structures is required. Such a structure could be represented with usual volume data, but the rendering performance would decrease and the memory consumption would increase without benefit, because the scene information is still sparse for a scene with thick material layers.

Potential of voxel representations

The general representation of complex and highly detailed scenes is one of the main goals in CG as well. Currently, voxel representations of a scene are one promising possibility to increase the level-of-detail for massive scenes by keeping the memory consumption moderate and the rendering performance high. Recent works like [LK10, CNLE09, KSA13, DKB*16] prove the benefit of using voxels as rendering primitives. They all have in common that the sparsity of a scene is exploited. Empty and filled spaces can be efficiently encoded in a hierarchical manner to store more detailed information of the scene geometry. Individual attributes per voxel allow a highly detailed scene appearance. Furthermore, sparse voxel representations have the advantage that geometry and appearance data are efficiently represented in a single structure. Hence, the use of

voxels is a very promising approach for an accurate THz simulation of highly detailed surfaces with thickness properties.

3.2 CONCEPT FOR THZ SIMULATIONS

Requirements for a data structure of a THz simulation

A THz simulation differs from common CG simulations of visible light due to different scattering behavior of the underlying radiation with respect to the radiated surfaces. While visible light has shorter wavelengths than the details of a real-world surface by orders of magnitude, THz radiation has similar wavelengths in comparison to the radiated surface structures. Therefore, the roughness of a surface becomes important, because wave effects contribute to the THz signal. Furthermore, active THz emission is commonly coherent, where visual light is mostly assumed as incoherent. Additionally, materials are penetrated by THz radiation so that inner structures need to be represented. If a simulation must consider these properties, the used representation for the scene must fulfill following requirements:

- representing highly detailed surfaces by increasing the scene resolution
- modeling geometrical surface properties like thickness or roughness explicitly
- efficient storage of material attributes
- allowing an efficient conversion of other scene representations, e.g. triangle meshes

[CNLE09] and [LK10] serve as basis for the SVT concept

Voxel structures are reasonable for representing scenes for a THz simulation in order to fulfill these requirements. The approaches of [LK10] and [CNLE09] propose efficient voxel structures and voxel rendering, so that an application of these approaches to a THz simulation is promising. Conceptually, both approaches can be applied to a THz simulation independently, but it is expected that a combination of both methods leads to a more suitable technique for the above mentioned THz requirements. In respect to memory efficiency, data creation, rendering properties and rendering efficiency, a discussion is provided in the following paragraphs to argue the resulting concept decisions for the SVT.

MEMORY EFFICIENCY The approaches of [LK10] and [CNLE09] provide voxel tree structures with very efficient memory consumptions. [LK10] optimizes the memory of an SVO by representing empty subtrees and empty voxels with single bits. [CNLE09] maintains subvolumes in a sparse N^3-tree structure. It allows to exploit the sparsity of the scene by removing empty subvolumes. The tree has the flexibility to adjust N, i.e. the children per node, which leads to a

reduced memory footprint due to a flattened hierarchy. In Sec. 3.3.1 both data structures are discussed in detail.

Decision for extending SVOs of [LK10]

For the representation of thick or rough layers in THz scenes, it is assumable that a combination of both techniques uses even less memory and a higher scene resolution can be obtained. Thick layers cover small subvolumes which would create a deeper hierarchy if only represented with an octree, but the overall sparsity of the scene is still high and large subvolumes would waste memory. Hence, the proposed SVT combines the octree encoding of [LK10] with the possibility to increase the number of children per node like [CNLE09] (see Sec. 3.3.2 and Sec. 3.3.4 for further details).

Constraints of volumes

Under the additional assumption that each voxel needs to hold a complex material description or a reference to an index of a material library, the attribute storage per voxel needs to be more flexible to allow an optimized memory footprint. General volume representations or the subvolumes of [CNLE09] store empty attributes less efficient, because a spatial sorting for interpolating the dataset is needed. Hence, the possibilities to use more complex attributes or to increase the scene resolution are constrained.

DATA CREATION As stated in [LK11], the input data of [LK10] is created by a voxelization of triangle meshes. Hence, no inner structure of the object is created. [CNLE09] uses meshes and volumes to create the scene representation. For volumes, existing datasets are copied or instantiated to obtain a higher resolution, but this approach does not allow to create a meaningful real-world scene at a high resolution. For meshes, noise functions are used to simulate more details. If explicit or locally varying surface characteristics need to be created, the application of noise functions becomes less practical because resulting patterns are repeated so that individual noise functions are required for different surface characteristics.

Decision for using unordered triangles sets

Although volume representations or direct voxel editing frameworks can be used for an explicit modeling of surfaces, it is more flexible and more convenient to create the voxel representation of a scene from surface meshes, because common 3D modeling tools allow an intuitive and comprehensible creation of surface structures. While it is possible to adjust geometrical features on the surfaces easily, the thickness property can only be obtained by creating several mesh layers or by using solid voxelization. If a solid voxelization is used, neighboring information between triangles or full access to the resulting voxel tree is required. Therefore, the possibilities for creating high resolution scenes from massive triangle data would be limited. Hence, an unordered set of triangles allows the maximum freedom to create high resolution representations with explicitly modeled surfaces. It follows that a voxelization in the spirit of [LK11]

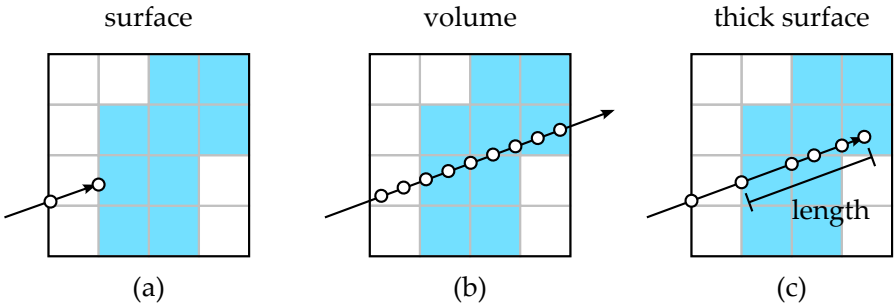


Figure 3.1: Different voxel rendering variants are depicted: A surface rendering like it is used by [LK10] is shown. The ray traversal stops at the first intersected voxel or a contour inside a voxel (a). If the voxel tree contains subvolumes like in [CNLE09], a volume traversal can be done by uniform sampling and trilinear interpolation (b). Continuing the ray traversal of the surface rendering after the first intersection leads to an adaptive sampling where the ray length inside a surface can be determined analytically (c).

with out-of-core properties is preferable. The proposed voxelization approach is discussed in Sec. 3.4.

RENDERING PROPERTIES Rendering properties are compared in Fig. 3.1. All methods use raycasting for traversing a voxel tree structure. The raycasting of [LK10] allows to render voxels or improved contours of the approximated mesh inside a voxel, but in both cases the traversal stops after the first intersection (see Fig. 3.1 (a)). No additional information of thickness or other surfaces behind is obtained. The surfaces which are penetrated by THz radiation cannot be determined. The volume rendering of [CNLE09] is done by trilinear interpolation and a common uniform sampling (see Fig. 3.1 (b)). Here, surface thickness could be obtained by determining isosurfaces.

Trilinear interpolation is impractical for attributes in lookup tables

The memory footprint of the data structure of [LK10] is more beneficial for storing thick surfaces, because it does not store empty voxel attributes. As a consequence, the spatial information is separated from the attribute information. This separation has the drawback that uniform sampling and trilinear interpolation get very inefficient and expensive, because each neighboring sample needs to be determined by traversing the voxel tree. An advantage of this separation is a more flexible memory requirement for attributes. Assuming that indices to a material library or compression is used, the overall memory consumption can be reduced, but trilinear interpolation would not be feasible, because lookup tables or compression schemes can only be resolved with a high computational effort. It follows that a common volume rendering of [CNLE09] is not applicable.

[LK10] allows adaptive and flexible sampling

To allow the determination of thick surfaces and a traversal of complex material descriptions still, the ray traversal of [LK10] just needs to continue after the first intersection (see Fig. 3.1 (c)). The advan-

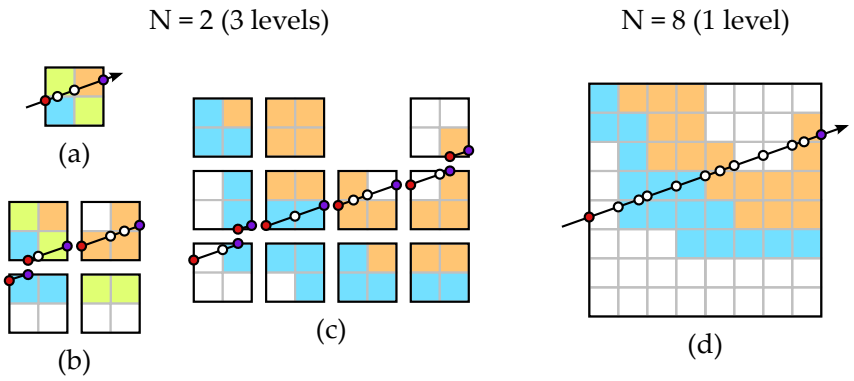


Figure 3.2: An example for estimating the sampling effort of an N^2-tree with varying N is shown. In terms of memory access and calculation, every switch between hierarchy levels with push- (red dots) and pop-operations (purple dots) is more expensive than step-operations (white dots) in the same hierarchy level. In the example of N=2, 3 levels from coarse to fine need to be traversed with 10 push-, 10 pop- and 10 step-operations (a-c). For N=8, 1 level needs to be traversed with 1 push-, 1 pop- and 10-step-operations (d). It is expected that the traversal of N=8 is faster than N=2 due to less switches between hierarchy levels.

tage is an adaptive sampling which allows material determination by lookup tables and an exact determination of ray lengths or surface thicknesses. A drawback for low scene resolutions becomes the resulting rendering quality, because the missing interpolation of sampled values leads to a less smooth appearance. The resulting lack of rendering quality can be compensated by creating the scene in a larger resolution which is obtained by optimizing the memory requirement and performance.

RENDERING EFFICIENCY Since [LK10] is optimized for surface rendering and [CNLE09] is optimized for volume rendering, a comparison of rendering efficiency is not meaningful and drawbacks can only be argued. [LK10] would be inefficient for volume rendering, because the octree has a deep hierarchy and each ray would traverse many hierarchy levels, which leads to a poor performance. If [CNLE09] would be used for surface rendering, it could be slower than [LK10], because many ray samples need to be taken to find the surfaces depending on the size of the subvolumes and the scene complexity.

Better performance is expected

The intended usage of the proposed SVT is neither surface rendering nor volume rendering. Therefore, it is difficult to estimate whether [LK10] or [CNLE09] is preferable. Theoretically, the influence of N needs to be considered, because the number of expensive switches between hierarchy levels and the number of necessary ray samples is reduced if N is increased. Both properties depend on the

scene material, because the THz radiation is not necessarily traversing the full scene. Hence, the choice of the optimal N is influenced by the scene content as well. If this scene dependency is neglected, it can be generalized that a shallow hierarchy and a reduced number of samples per ray lead to a higher performance, because less computation needs to be done (see Fig. 3.2). It follows that a performance gain is expected if an N^3-tree and an adaptive voxel sampling are used. Therefore, the proposed SVT allows to increase N and uses the adaptive traversal of [LK10]. The rendering of the proposed SVT is discussed in Sec. 3.5.

3.3 DATA STRUCTURE

3.3.1 *Related Work*

Octrees and its derivatives are well researched

First publications about the use of octrees in computer graphics appeared around 1980, e.g. [Mea82, Hun78] or [JT80]. Since then, many publications about octrees and its derivatives like 3D mipmaps ([LW90, LH91, BD02, DGPR02]) or N^3-trees ([CB04, LHN05, CNLE09]) appeared.

Current focus is on memory reduction and dynamic voxel data

Recent publications still improve on these voxel tree structures. [Mus13] focuses on level sets and dynamic volume representations which are stored in a hierarchical tree structure with similarities to B+Trees and very efficient bit encoding techniques to allow fast random access. While allowing a very efficient and performant processing of dynamic volume data, the memory requirement can not be as optimized as an SVO for surface rendering. [KSA13] propose a sparse voxel directed acyclic graph (SVDAG) to further improve the memory requirements of the SVO of [LK10] by collapsing subtrees with the same topology and transforming the tree to a graph. This graph does not allow to store material attributes. The most recent data structure of [DKB*16] removes this limitation and allows to store material attributes. The nodes of the SVDAG are extended by relative pointer offsets which allow a lookup of the corresponding material attribute. These pointer offsets are determined by a depth-first traversal of the SVO topology which is transformed to the SVDAG.

[CNLE09] and [LK10] are the two main influences

In respect to the proposed SVT, the two main influences are the SVO of [LK10] and the N^3-tree of [CNLE09] (see Sec. 3.2). Therefore, the data structures of those two approaches are discussed more in detail.

Efficient SVO data structure of [LK10]

An SVO structure with a very efficient memory footprint is proposed by [LK10]. The structure holds information about tree topology, shading attributes and voxel geometry. One main contribution is an efficient encoding of the topology. Fig. 3.3 shows an example. Instead of providing pointers to all child nodes of one parent, only one pointer to a memory region is used. In this memory region, all

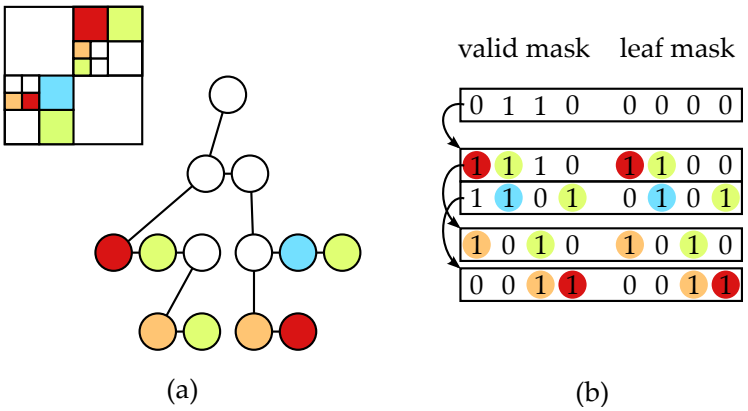
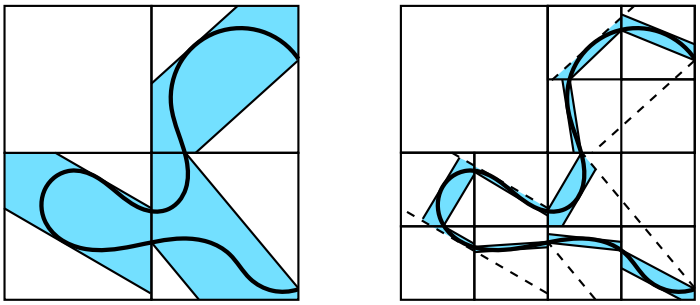


Figure 3.3: The SVO structure of [LK10] is shown as a 2D example. Only one pointer is used to address all child nodes of a parent (a). Storing the child nodes linearly in memory allows to jump to the correct child by the valid mask and obtain the information if the child is a leaf by the leaf mask (b).

non-empty child nodes are consecutively stored in memory. The correct child is found by calculating a pointer offset from a bitmask (see valid mask in Fig. 3.3 (b)). In the 3D case, this bitmask consists of 8 bits representing all possible children. If the respective bit is set, the child has an entry in the referenced memory region and can be determined by counting the set bits of the preceding child nodes. Another bitmask is used to define leaf-nodes, where each bit represents the state of one child again (see leaf mask in Fig. 3.3 (b)). This bitmask is used to stop branching of subtrees in homogeneous regions that do not need to be refined further. The memory consumption is reduced and the performance is improved due to a shortened tree traversal. Therefore, child nodes of one parent can have varying levels of detail. Another contribution is the approximative description of geometry inside the voxel with bounding slabs over the hierarchy levels. Two parallel planes enclosing the original mesh inside a voxel are stored in each node. These slabs are combined over the hierarchy levels to approximate the original mesh (see Fig. 3.4). If the resulting error between the original mesh and the approximative bounding planes is negligible, the nodes of the SVO do not need to be refined anymore at this position. Therefore, the memory consumption is reduced and the performance is improved by a reduced hierarchy depth. This technique is limited to a representation of surface models and is not applicable to volume representations.

N^3-trees of [LHN05] and [CNLE09]

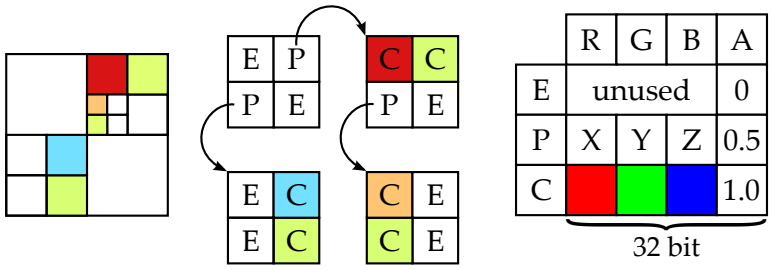
While the SVO of [LK10] is intended for surface rendering, the proposed structure of [CNLE09] is used to represent volume datasets. Thus, the design of the tree structure has other requirements, because the memory consumption becomes higher and the rendering needs to be done in a more performant manner to achieve similar framer-



(a) (b)

Figure 3.4: The surface approximation with hierarchical slabs of [LK10] stores parallel planes by a normal and two distances in each hierarchy level to approximate the surface and to stop the refinement of the voxel tree early. The information of the coarser levels (a) is used to calculate a tighter voxel slab in the finer level (b).

ates. To address these requirements, [CNLE09] uses an N^3-tree and extends it with the property that each node can have a pointer to a 3D-texture. This texture represents the corresponding subvolume of the dataset. The N^3-tree implementation is similar to the structure of [LHN05] (see Fig. 3.5) and allows a shallow hierarchy representation which is more suitable for volume information. The GPU texture memory is used to store the nodes next to each other. [LHN05] uses RGBA8 channels as one pointer to all child nodes or as color values for texturing. [CNLE09] stores nodes with 64 bits (see Fig. 3.6). One bit defines if the original volume data set still contains data and can be refined further. 30 bits represent a pointer to child nodes. One RGBA8 color value with 32 bits is used for the representation of the



(a) (b) (c)

Figure 3.5: The N^3-tree structure of [LHN05] is depicted. For simplification a quadtree (2^2-tree) is shown. The original tree (a) is represented by pixels in texture memory (b). All child nodes of one parent represent a contiguous block in texture memory and can be addressed by one texture index. The RGB channels of a 32 bit-pixel (c) are either empty nodes E, interpreted as a pointer P to all child nodes or used as a color C. The 8 bits of the alpha channel are used to interpret the pixel as one of these three possibilities.

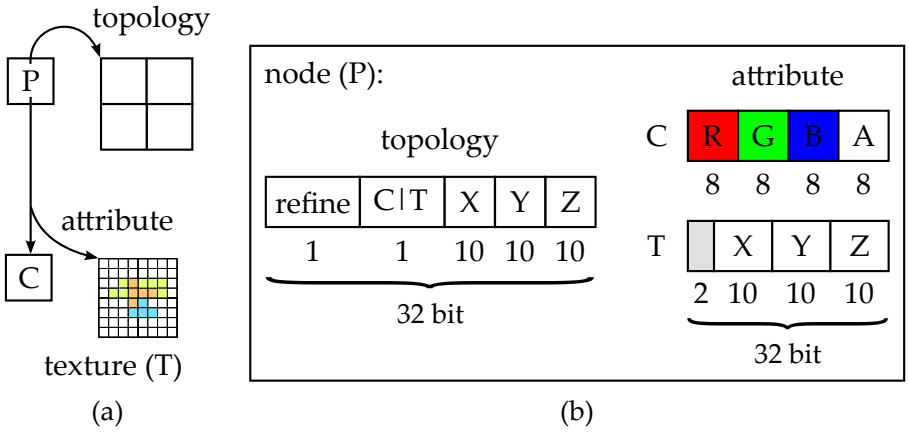


Figure 3.6: The node representation of [CNLE09] is shown. A node P references topology and attributes (a). While the topology is always an index to a all child nodes in texture memory, a color C or a link to another texture T represent the attribute (b). C represents an empty or homogeneous subvolume. T is a texture that represents a heterogeneous subvolume. T can have another resolution M^3 than the resolution N^3 of the voxel tree.

subvolume. One bit is used to decide if this color value either is interpreted as a homogeneous region or as a pointer to another region of texture memory. The referenced texture holds information about the subvolume and can have another resolution, i.e. the tree has nodes with N^3 elements and the nodes redirect to other texture blocks with a resolution of M^3. With this flexibility it is possible to represent adaptive volume datasets efficiently.

3.3.2 *Generalization of SVOs to SVTs*

The proposed SVT generalizes the SVO of [LK10] to an N^3-tree.

The proposed SVT of this thesis generalizes the SVO structure of [LK10], which is used for memory-efficient and highly performant surface rendering, to a N^3-tree in the spirit of [CNLE09], which is more suitable for volume-rendering but wastes more memory for empty space in sparse scenes (see Sec. 3.3.1). Fig. 3.7 gives an overview of the generalization concept. Like the SVO of [LK10], the SVT stores one pointer to all child nodes of one parent. All child nodes of a parent are stored linearly and a bitmask is used to address the correct node. The main difference between SVTs and SVOs lies in the number of these addressable child nodes which are linearly stored in consecutive memory. The SVO can only store up to 8 child nodes. In theory, the SVT can store all child nodes of the whole voxel grid with only one pointer and one bitmask (see Fig. 3.7 (b)), but the implemented SVT limits the number of child nodes to 125 due to memory restrictions and performance losses in the rendering for more child nodes (see Sec. 3.3.4). Furthermore, Fig. 3.7 (a) shows the

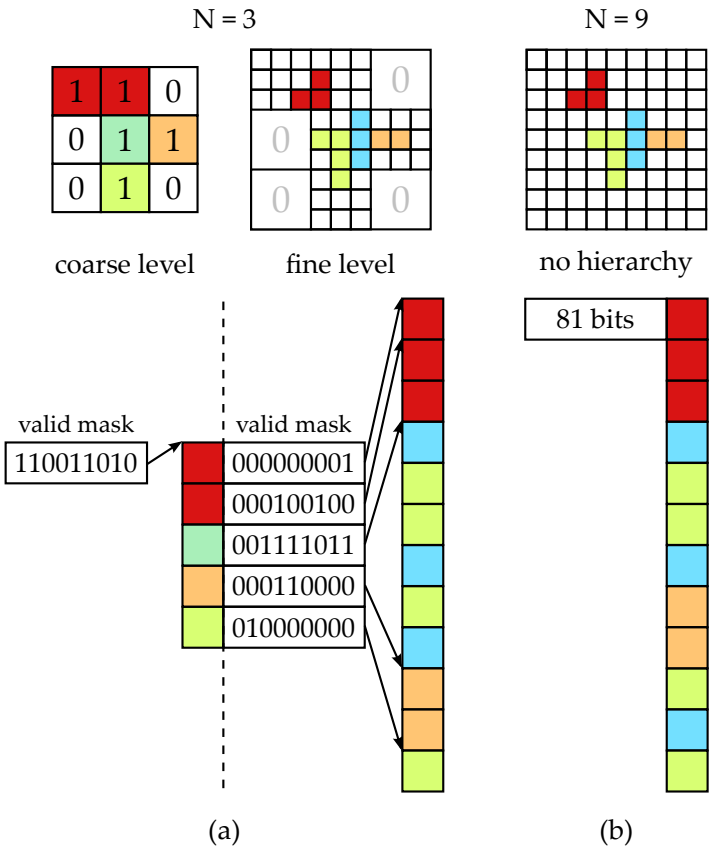


Figure 3.7: The SVO of [LK10] can be generalized to SVTs if the number of child nodes can vary. While the SVO has a fixed number of 8 children per node. The SVT has N^3 children per node. Examples for N = 3 (a) and N = 9 (b) are given. The tree and the corresponding data structure are compared. For N = 3, two hierarchy levels are needed. The corresponding data structure efficiently encodes subspaces by single bits and stores single pointers to all child nodes which are consecutively stored in memory (a). In the theoretical case when N has the same length as the full voxel grid (b), no pointers are needed but empty voxels need one bit on the finest level.

efficient encoding of empty voxels in a hierarchical representation. In comparison to the SVO, larger groups of empty child nodes can be encoded by single bits in the coarser levels of the hierarchy, if the subspace is empty.

Key technique for generalization is the generalized Morton order

In general, the needed sorting of child nodes is obtained by the Morton order [Mor66] which maps multidimensional data to a linear index. This order preserves locality, because contiguous Morton indices correspond to a group of child nodes which belong to the same parent node in the next coarser hierarchy level. The Morton order is calculated by an interleaving of bit representations of the coordinates which need to be transformed. Since bit representations use base 2, the ordering exactly represents the structure of an octree, be-

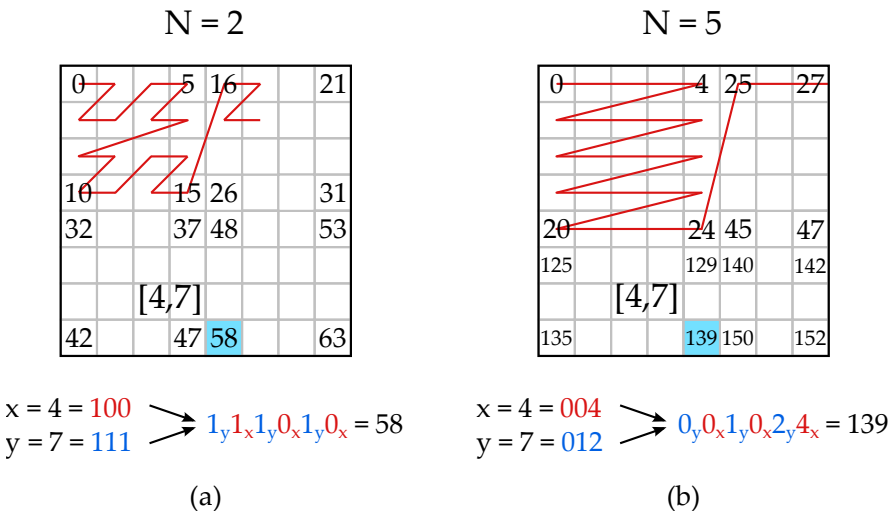


Figure 3.8: The exemplary calculation of the Morton code is shown. It is the basis for a linear sorting of child nodes of an SVT, because groups of consecutive Morton numbers represent child nodes that belong to the same parent node. While the common Morton calculation on base 2 is used for SVOs (a), the SVT requires a generalized calculation on the desired base N of the N^3-tree (b).

cause only one subdivision in each dimension is used. For SVTs, this principle of Morton indexing needs to be extended to a larger base, i.e. the coordinates are transformed to its representations on base N, interleaved and transformed back to decimal base (see Eq. 3.1 and 3.2). Fig. 3.8 (a) shows the common calculation of Morton indices. As an example for the generalized method, Fig. 3.8 (b) shows the Morton calculation on base N = 5.

$$\left.\begin{aligned} x &= x_k \cdot N^k + \ldots + x_1 \cdot N + x_0 \\ y &= x_k \cdot N^k + \ldots + x_1 \cdot N + x_0 \end{aligned}\right\} y_k x_k .. y_1 x_1 y_0 x_0 \tag{3.1}$$

$$index = y_k \cdot N^{k \cdot 2+1} + \ldots + y_1 \cdot N^3 + x_1 \cdot N^2 + y_0 \cdot N + x_0 \cdot 1 \tag{3.2}$$

Difference between SVT and N^3-tree

[CNLE09] and [LHN05] create generalized N^3-trees and store single pointers for referencing all child nodes of one parent as well. In comparison to these approaches, the SVT does not use texture blocks to maintain all child nodes that a parent can have. Attributes are only stored for valid voxels, while empty voxels are only represented by single bits in the bitmask. This method reduces the memory consumption of scenes with many empty regions, i.e. sparse datasets are stored more efficiently.

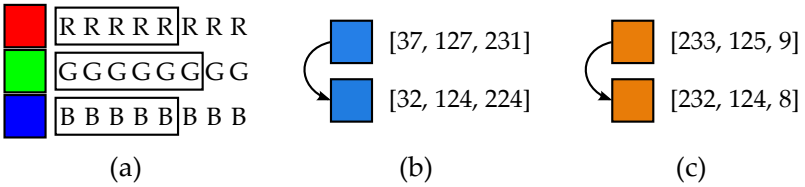


Figure 3.9: The RGB565 compression is used to store color values in the SVT. If an RGB8 color is given as input, the least significant bits of the R, G and B channel are removed (a). Two examples of the resulting quantization are shown (b,c).

3.3.3 *Attribute Compression*

*RGB565 and octahedron compression of [CDE*14] are used*

In addition to topology information, all voxel tree approaches need to store attributes. The implemented SVT stores color and normals for each node. The color is compressed by the RGB565 format, which needs 16 bit for the color representation (see Fig. 3.9). This compression scheme uses 5, 6 and 5 bits for the red, green and blue color channel, respectively. It is obtained by removing the least significant bits of usual 8 bit RGB representations. The normal information is encoded into 16 bit as well. The octahedron encoding of [CDE*14] is used. Here, the normal direction on a sphere is projected to an octahedron. By calculating an unfolding of this octahedron to a quad, the normals $\vec{n}$ can be represented by texture-coordinates $[u, v] \in [-1, 1]^2$ with a precision of 8 bit per coordinate. Fig. 3.10 illustrates the approach. Eq. 3.3 is used for normal-encoding, while Eq. 3.4, 3.5 and 3.6 are used for normal-decoding.

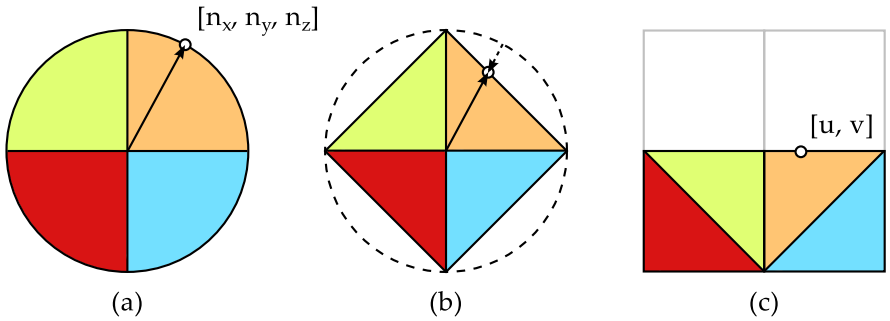


Figure 3.10: The octahedron encoding of [CDE*14] allows to store compressed normals in 16 bits. After a projection of the normal $\vec{n}$ (a) to an octahedron (b), the octahedron is unfolded to an image and the coordinates of $\vec{n}$ are transformed to [u,v]-coordinates (c).

$$[u,v] = \begin{cases} \frac{[n_x,n_y]}{|n_x|+|n_y|+|n_z|} & \text{if } n_z > 0 \\ 1 - |\frac{[n_y,n_x]}{|n_x|+|n_y|+|n_z|}| \cdot sign([n_x,n_y]) & \text{if } n_z \leqslant 0 \end{cases} \tag{3.3}$$

$$n'_z = 1 - |u| - |v| \qquad \text{with } [u,v] \in [-1,1]^2 \tag{3.4}$$

$$[n'_x, n'_y] = \begin{cases} [u,v] & \text{if } n'_z > 0 \\ 1 - |[v,u]| \cdot sign([u,v]) & \text{if } n'_z \leqslant 0 \end{cases} \tag{3.5}$$

$$\vec{n} = \frac{\vec{n'}}{\|\vec{n'}\|} \tag{3.6}$$

with $sign(x) = 1$ for $x \geqslant 0$ and $sign(x) = -1$ for $x < 0$

Comparison with compression methods of related approaches

Compared to [LK10], the SVT does not store contours for representing voxel geometry. Furthermore, the SVO encodes color and normal information in a different manner. The color is encoded with a simplified variant of DXT1 compression (see [vW06] and [INH99]), while a new normal compression is proposed (see [LK10] and Sec. 3.5.4.3). [CNLE09] and [LHN05] do not use compression for storing attributes.

3.3.4 *Memory Consumption*

Memory layout of the SVT and limitation to N = 5

The implemented SVT structure consists of 3 memory regions (see Fig. 3.11). The first region stores the bitmasks for encoding the topology, the second region represents the pointers to the child nodes and the third region stores corresponding attributes of the node. The memory regions are stored in separate textures for an efficient lookup in the rendering. Currently, a maximum of 128 bits can be obtained with one texture-lookup by hardware. Therefore, the SVT implementation supports N^3-trees with $N = \{2,3,4,5\}$ because the bitmask for representing the topology with $N = 5$ needs 125 bits. An SVT with $N = 6$ would exceed the accessible 128 bits and an additional texture lookup for every node would be necessary. Other bit operations, which are needed for a traversal of the SVT, like counting or shifting are limited to 64 bit on current GPUs. Therefore, the processing of the proposed SVTs with $N > 5$ would become impractical and inefficient. Independent of N, the other two memory regions, i.e. child pointers and attributes, both use 32 bit to store one element.

Memory footprint of single nodes

Bitmasks and child pointers are used to reference the child nodes of a node. Therefore, the finest level of the hierarchy does not need them. Only the attributes are stored on all levels. It follows, that the leaf nodes have a memory footprint of 32 bits. The theoretical footprint of an inner node consists of 32 bits for an attribute, 32 bits for a child pointer and N^3 bits for the bitmask. Since the bitmask are

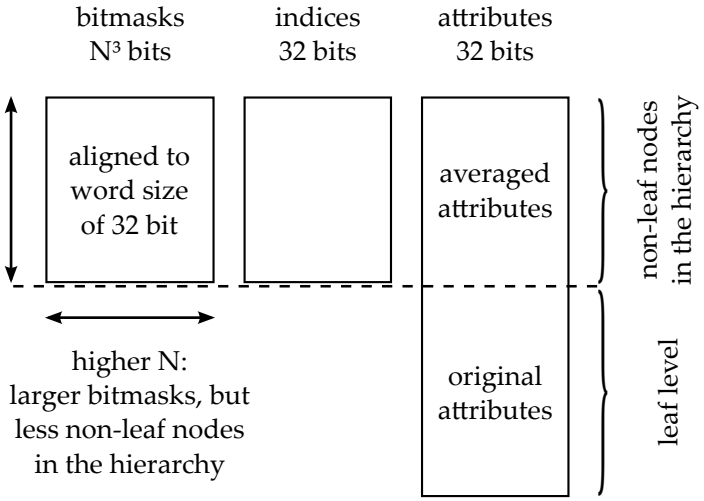


Figure 3.11: The structure of the SVT is stored in three textures: 1) Each node in the hierarchy has an attribute. The leaf attributes are determined from the original data set. The parent nodes of the coarser levels combine the attributes of its child nodes, e.g. the attributes are averaged. 2) Each node has an index, which points to its first child node. 3) A bitmask is stored for each node to address the correct child by bit counting after jumping to the first child node. The bitmasks are aligned to a word size of 32 bit, because RGBA32 channels of a texture are used - Changing N leads to a reduced hierarchy depth, but a larger number of bits per bitmask.

aligned to 32 bits for color channels of a texture, the footprints are as follows: $N = \{2, 3\} : 96$ bits, $N = 4 : 128$ bits, $N = 5 : 192$ bits.

Memory requirement of the scene

The total memory consumption of a scene is influenced by the number of nodes in the hierarchy and the complexity of the scene. A higher hierarchy with $N = 2$ has more nodes with smaller footprints, while a reduced hierarchy depth with $N = 5$ has less nodes with larger footprints.

Example for the choice of N

Depending on the scene composition and the relation of bits for representing empty space and attributes which need to be stored, a larger N optimizes the memory requirement of the topology. If the bits in the bitmask exceed the bits used for attributes, a larger N needs more memory again. Fig. 3.12 shows a simplified example of an SVT in 2D. On the one hand, more attributes need to be stored for N = 2, because a hierarchy is necessary. On the other hand, more bits in the bitmask need to be stored for N = 8. It follows, that N = 4 has the lowest memory consumption in this case. Table 3.1 gives a detailed information about memory consumption for that example. An evaluation of the influence of N on the memory consumption for example scenes in Sec. 3.4.6.2 shows that the optimal N is dependent on scene sparsity and implementation details. Therefore, the optimal

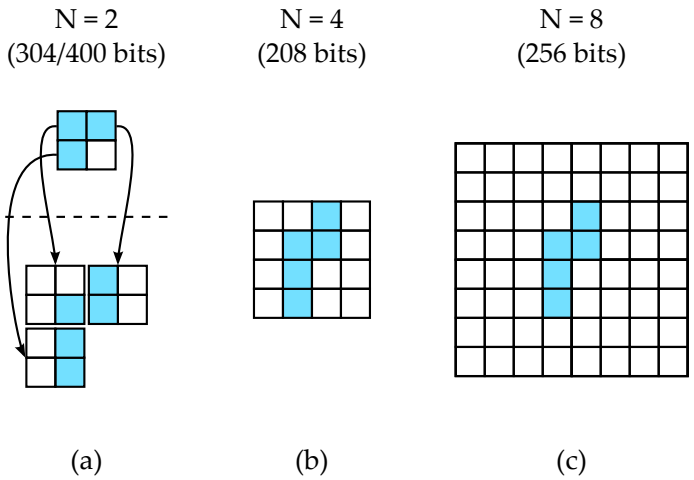


Figure 3.12: The memory consumption of the hierarchy is influenced by the choice of N for the SVT. The same scene is represented with N = 2 (a), N = 4 (b) and N = 9 (c). While N = 4 has the optimal memory consumption, the SVT with N = 8 needs to store more empty bits in the topology and the SVT with N = 2 needs to store more topology information.

value of N is not constant and needs to be adjusted to achieve the optimal memory consumption of the SVT for a specific scene.

Memory consumption of leaf nodes is not influenced by N

The main part of the memory is used by the attributes in the leaf nodes. Here, the needed memory is independent of N. Therefore, a larger N improves the memory which is needed for topology but the memory consumption of the attributes can only be improved by more efficient compression methods.

N	Bitmask	Indices	Attributes	Total
2	$4 \cdot 4$	$4 \cdot 32$	$(5\|8) \cdot 32$	304|400
4	16	32	$5 \cdot 32$	208
8	64	32	$5 \cdot 32$	256

Table 3.1: Memory consumption in bits for the example given in Fig. 3.12. To provide a fairer comparison, a variant with attributes in the coarse hierarchy level and a variant without attributes in the coarser hierarchy level is provided for N = 2. If the used word alignment of 32 bits (see Fig. 3.11) is considered, the two variants of N = 2 would have 416 or 512 total bits and the SVT with N = 4 would have 224 total bits. It would not change for N = 8.

3.4 VOXELIZATION

3.4.1 *Overview*

Short description of approach, contributions and difference to [PK15]

The following section is based on [PK15]. It presents an out-of-core construction of SVOs on GPUs to allow a very performant processing of massive triangle meshes which do not fit into GPU memory, while allowing a sophisticated voxel attribute creation without constraints on the order of contributing attribute values per voxel. A streaming approach for triangles and voxels allows to process only subsets of the whole dataset. Triangle subsets can be voxelized individually. Afterwards, the resulting intermediate subset of voxels can be used to create subparts of the SVO and built the SVO sequentially. With a generalized calculation of the Morton index (see Sec. 3.3.2), a construction of SVTs is achieved with the same method. The main contributions of the proposed approach are:

- A grid-free voxel handling which allows an out-of-core voxelization on GPU and removes the need of processing empty voxels,
- Parallel voxelization of triangles on GPU which is optimized for a balanced workload, avoiding any atomic operations as usually used in grid-based parallel triangle voxelization approaches
- Maintenance of all triangle properties on a per-voxel level to allow an attribute determination in post-order for the voxel and the resulting nodes of the SVT, and
- The possibility to create a generalized N^3-tree structure with an improved memory efficiency in comparison to current SVO implementations.

3.4.2 *Related Work*

Focus on SVO creation

The voxelization of SVTs or SVOs differs in many aspects from usual voxelization methods due to the fact that empty voxels are usually omitted and not stored in the data structure which will be created, i.e. sparse tree structures will be generated. Therefore, the focus of related work lies on methods which create SVOs as well. A recent in-depth discussion about voxelization in general is given in [Lai13].

In-Core GPU

A surface and a solid voxelization on the full grid is presented by [SS10]. Furthermore, [SS10] present a solid voxelization method to an SVO representation. In all cases, the GPU rasterizer is not utilized. The sparsity in the voxelization is exploited by determining and processing lists of active nodes in the octree construction. The octree structure is created in a bottom-up fashion starting from the parent nodes of the finest level to fill the leaf nodes with the voxel data afterwards. Therefore, the octree structure and triangle data need to

be stored in memory completely. The attributes of the generated voxels only hold binary information or a scalar representing the coverage factor, while the proposed voxelization stores voxel color and normal.

[CG12] proposes a sparse voxelization on GPU that uses the hardware rasterizer. A conservative rasterization is used to generate the voxels. The tree generation is done top-down. All octree-nodes, which contain the generated voxels of the finest level, are determined. The resulting nodes on leaf-level are filled with the attributes from a voxel fragment list and coarser levels get averaged attributes from the child nodes bottom-up. The available GPU memory for storing the complete voxel fragment list and the resulting octree-structure limits the size of the scene representation.

In-Core CPU

A detailed introduction to SVOs is given in [LK11]. A top-down approach on CPU determines independent slices containing parent-child relations and the needed triangles for the subtree to generate the SVO. This method would allow an out-of-core SVO creation, but especially the root node would need to contain all triangles for processing. Therefore, the SVO generation is limited to the number of triangles and the size of the created SVO that can fit into memory. A unique characteristic to other voxelization techniques is the triangle processing on every level of the tree. Here, the original input is used on all levels for attribute creation. By using the triangle data on each level, [LK11] proposes a contour creation and an early termination of subtrees in the voxelization, if these contours approximate the triangle geometries well enough. (see Sec. 3.3.1)

[KSA13] improves the memory usage of SVOs by storing them as a directed acyclic graph (DAG). Therefore, larger SVOs can be represented. To avoid a construction of the complete SVO at once, subtrees of the SVO are transformed into sub-DAGs. Afterwards, the final DAG is obtained by assembling the sub-DAGs. The tree construction has the same limitations in respect to out-of-core processing as [LK11], because all triangles, the intermediate SVO, and the final DAG need to fit into memory.

Out-of-Core CPU

The out-of-core CPU method of [BLD13] uses an efficient streaming for SVO creation and voxelization. First, the final grid is partitioned into smaller subgrids that fit into memory. Then, all subgrids are sequentially processed by voxelizing the stream of triangles and creating the corresponding subtree of the SVO. To remove the need for processing a large number of empty subtrees, an optimized SVO creation is proposed.

In a subsequent work, [BLD14] suggests the direct use of a voxel list, which improves the performance, but does not remove the need of maintaining the full grid, because there is no estimation for the possible size of the created voxel list in a subgrid. The voxel attributes are created independently from the triangle position, i.e. from the voxel's barycentric coordinates with respect to the triangle. Furthermore, the

first triangle intersecting the voxel generates the attribute. Extending this approach to allow arbitrary attribute computation would require the maintenance of all triangle contributions per voxel and functionalities like texture lookups need to be re-implemented on CPU. A significant performance loss would be the result.

Differences to [SS10], [BLD13] and [BLD14]

The proposed approach of this section is similar to [BLD13, BLD14], because triangle and voxel streaming is used as well. However, all triangle attributes per voxel are accessible for computing the voxel attributes in the proposed approach. Furthermore, triangle sorting replaces the subgrid-partitioning in order to be more flexible and grid-free. In contrast to [BLD13, BLD14], which uses the surface voxelization of [SS10], the conservative voxelization of [SS10] is used in the proposed approach. Additionally, the voxelization is done on GPU rather than CPU. In comparison to [SS10], the proposed approach creates leaf voxels directly and creates SVT parts bottom-up in a single pass to enable out-of-core processing.

3.4.3 *Algorithm Overview*

Approach consists of triangle and voxel stream processing

The proposed construction of SVTs comprises the stream processing of triangles and voxels. The main goal of the triangle stream processing is to prepare a given triangle set $\{T_i\}$ in such a way, that parallelized processing can efficiently be performed on GPU, while the voxel stream processing is intended to maintain voxels and construct the SVT in a sequential order. Figure 3.13 gives a schematic overview of both processing stages, while a brief description is given in the following.

Short description of triangle stream processing

- Triangle stream processing:

  First, a subdivision scheme is applied on the input triangles. It improves spatial locality of the triangles, voxel handling efficiency and processing performance on GPU (see Sec. 3.4.4.1).

  Next, the triangles are sorted to achieve a voxel prediction, which determines the position of the triangle in a sorted set for the SVT creation and estimates how many triangles per iteration can be processed in parallel (see Sec. 3.4.4.2). For describing these triangle sets per iteration, the term *(triangle) batch* is used in the following.

  When the triangle batches are defined, a sequential processing of batches starts the voxelization and attribute creation (see Sec. 3.4.4.3). From the voxelization of one batch, a *voxel attribute set* is created. This set is a list of voxel-attribute pairs, which contains non-unique Morton indices and attribute values of triangles at specific voxel positions. Morton indices [Mor66] are used to represent the voxel indices, because they allow to map multidimensional data to a linear index that preserves locality.

This property is needed for the creation of N^3-tree structures like the SVT (see Sec. 3.3.2), because all children of one parent can be accessed by a contiguous range of indices.

Short description of voxel stream processing

- Voxel stream processing:

  To avoid the creation of SVT nodes that can be intersected by subsequent triangles, voxels with potentially incomplete information for attribute determination need to be maintained. For this purpose, a *Morton queue* is used (see Sec. 3.4.5.1). It stores all unprocessed voxel-attribute pairs of previous batches and all pairs of voxel-attributes of the current batch to extract voxels that can be used for SVT creation in the current iteration. The resulting voxel-attribute pairs represent the current voxel attribute set until the next triangle batch is voxelized.

  The SVT Creation is the second step (see Sec. 3.4.5.2). New parts of the SVT are built hierarchically in a bottom-up manner by starting with the leaf nodes. Attributes of the leaf nodes are calculated with the current voxel attribute set. On coarser hierarchy levels, nodes are only created if the complete attribute information of their descendants is available. Therefore, an additional *stitch queue buffer* is used to maintain the advancing front of incomplete nodes in the hierarchy. These nodes are finalized by subsequent batches. On CPU, the creation of the SVT is finished. The remaining nodes in the stitch queue buffer are iterated after all batches have been processed.

3.4.4 *Triangle Stream Processing*

3.4.4.1 *Subdivision of Triangles*

A triangle subdivision is needed for balancing GPU workload

The first step of the proposed algorithm consists of pre-processing. It consumes triangles sequentially and applies a subdivision scheme. The main goal is to create triangles which are as homogeneous as possible with respect to the following criteria:

- Among all triangles, the number of voxels generated per triangle $N_{vox/tri}(T_i)$ should be as equal as possible
- The range of Morton indices m, which are generated for each voxel of a triangle, determines how long a triangle stays active in the out-of-core scheme. Therefore, the Morton indices should be as compact as possible, i.e.

$$(m^{max}(T_i) - m^{min}(T_i))/N_{vox/tri}(T_i) \rightarrow \min!,$$

  where $m^{min}(T_i)$ and $m^{max}(T_i)$ denote the minimum and maximal Morton index generated by triangle T_i, respectively.

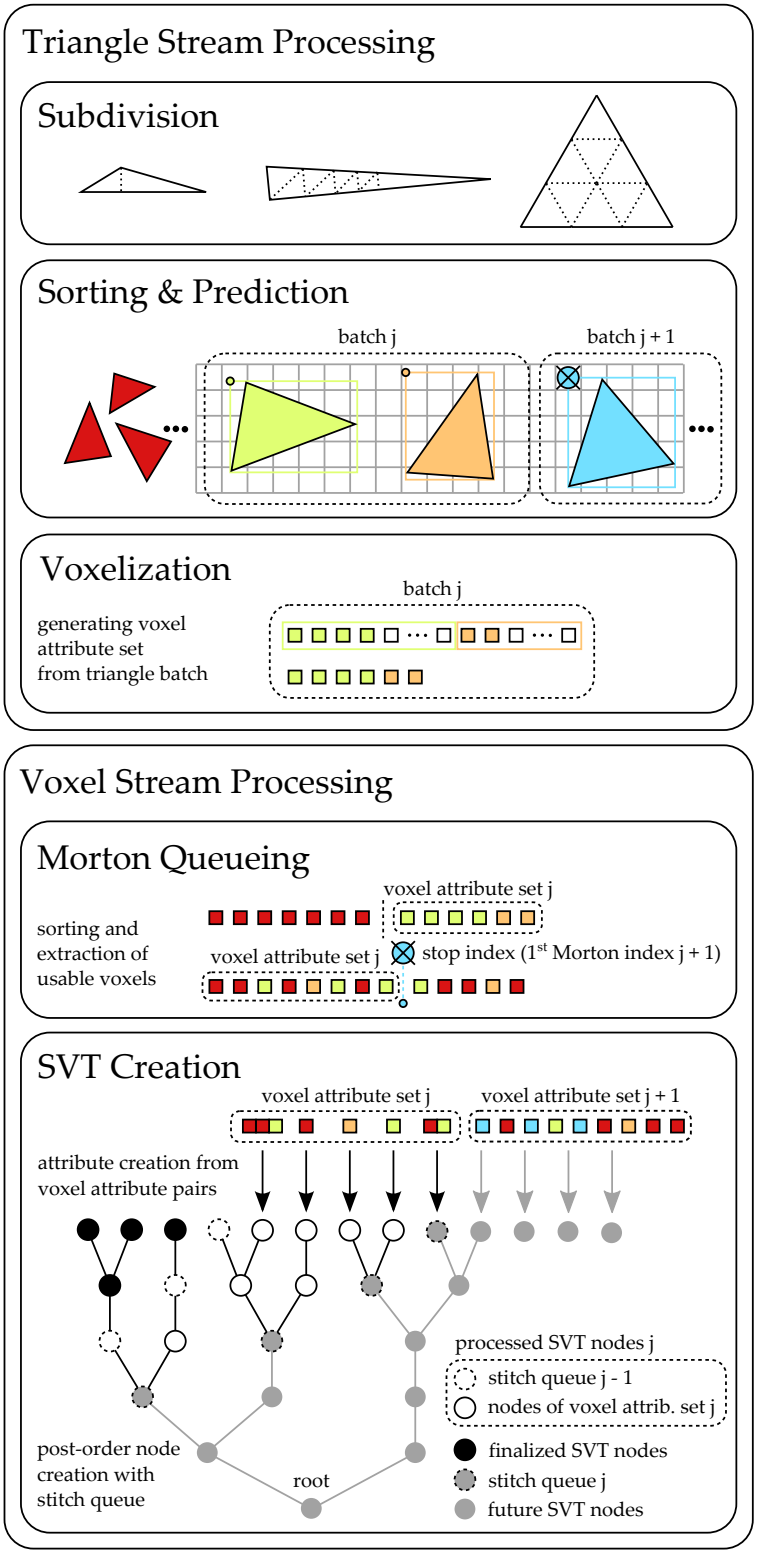


Figure 3.13: Overview of the proposed approach, which shows the steps of triangle stream processing and voxel stream processing. Image source: [PK15]

Of course, the simple solution of one voxel per triangle has to be avoided, i.e. $N_{vox/tri}(T_i)$ should not be too small, because the I/O time for transferring a higher triangle count would increase while the computation time for processing less voxels per thread would decrease. Thus, the GPU processing would not improve the performance of the voxelization.

Purely geometric subdivision rules are applied

Both criteria are influenced by the shape and size of the triangles. In addition, the specific spatial location of the triangle influences the Morton-range criterion strongly, but a determination of compacted Morton ranges would basically require voxelization already. Therefore, both criteria are approximated by using purely geometric subdivision rules. While a large Morton range for a single triangle can not be avoided by these rules, the subdivision limits the number of resulting voxels that need to be maintained during batch processing.

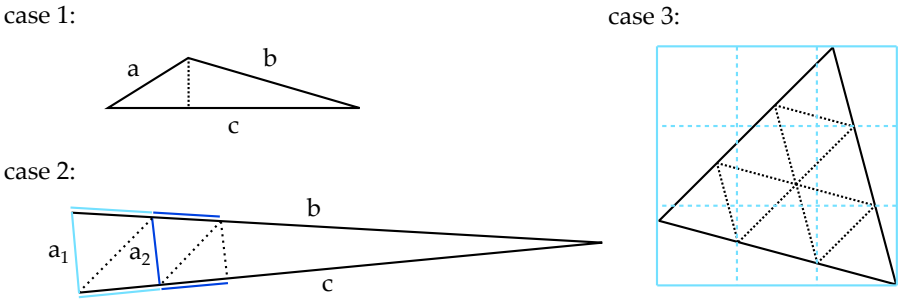


Figure 3.14: Triangle subdivision: The two cases of elongated triangles (case 1 and 2) and the regular case 3. Image source: [PK15]

Estimation of voxels per triangle

The criterion of compact Morton-ranges is solely addressed by preventing long and thin triangles (see Fig. 3.14 case 1 and case 2). In order to estimate the number of created voxels for triangle T_i, a canonical approach would be a projection of the triangle along the dominant coordinate axis of its normal. This projection relates the projected triangle size $A_{proj}(T_i)$ to the area of a voxel face A_{vox}. However, experiments have shown, that it is advantageous to use the area of the projected bounding box of the triangle $A_{bbox}(T_i)$, because long thin triangles are penalized additionally. Therefore, the final subdivision rule is given by Eq. 3.7, where $K^{max}_{vox/tri}$ is a user-defined parameter. The triangle is subdivided, if $A_{bbox}(T_i)$ does not fulfill Eq. 3.7.

$$A_{bbox}(T_i) \leqslant K^{max}_{vox/tri} \cdot A_{vox}, \tag{3.7}$$

The subdivision rules are distinguished between 3 cases

All triangles are checked for three different cases in a sequential process on CPU. These cases are depicted in Fig. 3.14 (see also Alg. 3.1). Under the condition that the three edges of triangle T are sorted according to their lengths $T.a \leqslant T.b \leqslant T.c$, case 1, i.e. two short and one long edge, and case 2, i.e. two long and one short edge, are determined by $T.a \leqslant T.b \ll T.c$ and $T.a \ll T.b \leqslant T.c$, respectively.

Algorithm 3.1 : Triangle subdivision. Source: [PK15]

```
// Input
{T_i}                    // triangle soup
K^max_vox/tri            // max. voxels per triangle
L                        // length of a voxel
A_vox ← L^2              // area of a voxelface
Q ← {T_i}                // Initialize input queue
R ← ∅                    // Initialize result queue

// Process the triangle soup
while Q ≠ ∅ do
    T ← Q.dequeue()
    // Apply subdivision scheme if area of bbox too large
    if A_bbox(T) ⩽ K^max_vox/tri · A_vox then
        // Case 1: T.a ⩽ T.b ≪ T.c, angles larger ~ 90°(√2)
        if (T.a + T.b) < √2 · T.c then
            Q.append(T.split())
        // Case 2: T.a ≪ T.b ⩽ T.c, angles smaller ~ 20°(3)
        else if 3 · T.a < T.b then
            Q.enqueue(T.subdivide())
        // Case 3: regular split
        else
            R.enqueue(T.regSubdiv())
{T_i} ← R // Resulting set of triangles satisfies Eq. 1.
```

First, all triangles are stored in a queue Q. Case 1 is handled by simply splitting the triangle in two parts, resulting in cases 2 or 3. In case 2, approximately equilateral triangles are created. Here, the short edge length $T.a$ is cut off the long edges and the resulting quadrilateral is split into two triangles. Until the remaining part of the triangle fulfills Eq. 3.7, this process is repeated. In case 3, the triangle is directly subdivided by simply computing $\left\lceil \sqrt{A_{\text{bbox}}(T)/(K^{\text{max}}_{\text{vox/tri}} \cdot A_{\text{vox}})} \right\rceil$ as subdivision factor. The created and the remaining triangles fulfill Eq. 3.7 after handling the subdivision cases.

3.4.4.2 *Triangle Sorting and Voxel Count Prediction*

Batch processing is needed for out-of-core voxelization

It is assumed, that SVO and triangle data do not completely fit into memory for global processing. Hence, the triangles need to be structured to allow the processing of triangle subsets, i.e. batches of triangles, and respective subtrees of the SVO. Since batches are processed sequentially, it is required to have all relevant SVO data on GPU for a subtree creation.

The triangles need to be prepared for two main challenges in the Voxel Stream Processing (see Sec. 3.4.5):

- Since the triangles in each batch $B_j = \{T_{j_{min}}, \ldots, T_{j_{max}}\}$ are processed without atomic operations and in parallel, an upper bound $N^{max}_{vox/tri}(T_i)$ of voxels generated by triangle T_i is needed. This bound allows to pre-allocate the memory for batch B_j and to determine write-offsets in a global memory for each triangle T_i.
- It is necessary to know, which voxels can be completed in the current triangle batch, because the finalized subtree of the SVO needs to be streamed out in order to avoid a memory overflow.

Triangle sorting with Morton indices

The properties of the Morton order [Mor66] are used for the sorting of triangles. It provides a linear order as if an SVT is traversed in depth-first post-order (see Sec. 3.3.2). [LGS*09] uses the same method of sorting primitives in the context of Bounding Volume Hierarchy (BVH) construction. While [LGS*09] takes the barycenter of the triangle bounding box as representative position of the triangle, the proposed approach requires the minimal Morton index $m^{min}(T_i)$ of triangle T_i, because this indicates that the voxel with Morton index $m^{min}(T_i) - 1$ is finalized after processing all triangles $T_1, \ldots, T_{i-1}$. As stated already, computing $m^{min}(T_i)$ would hardly be possible without voxelizing the triangle at this stage. Therefore, the bounding boxes of the triangles are used to calculate the smallest Morton index $\widetilde{m}^{min}(T_i)$ as lower bound of $m^{min}(T_i)$. Here, $\widetilde{m}^{min}(T_i)$ is calculated from the point of each bounding box that has the minimum Euclidean distance to the origin of the voxel grid which encloses all triangles of the scene. Furthermore, this index determines the earliest possible point in time when the triangle must be available for voxelization. Hence, the triangles are sorted to $\widetilde{m}^{min}(T_i)$ and $\widetilde{m}^{min}(T_{j_{min}})$ is stored as a *stop index*. This index of the first triangle of each batch can be interpreted as a stopping criterion for the extraction of a voxel attribute set from the Morton queue (cf. Sec. 3.4.5.1), because it provides the first voxel of the prior batch that cannot be used for node creation of the SVT.

Maximum number of voxels per triangle is estimated

In order to determine the upper bound $N^{max}_{vox/tri}(T_i)$ for the generated voxels of triangle T_i, the conservative voxelization of [SS10], which is applied further, needs to be considered. Each triangle is projected along the dominant axis direction of the triangle normal in [SS10]. In the proposed approach, the projected bounding box of the triangle is discretized to the voxel resolution and the count of enclosed projected voxels is determined. A maximum of three voxels along the projection direction will be created, because the triangle is planar [SS10], i.e. $N^{max}_{vox/tri}(T_i)$ is given by the "planar" voxel count per triangle multiplied by the factor of three.

Influence of number of voxels per batch

Naively, one would try to process the largest batches possible, but the "optimal" batch-size is very hard to determine and strongly depends on the specific dataset. A poor utilization of the GPU processing power is the result, if very small batches are processed. Large batches may lead to a "fragmentation" of the Morton queue (see

Algorithm 3.2 : Predict voxel counts, sort triangles & create batches. Source: [PK15]

```
// Input
  {T_i}              // triangle soup
  Vbox               // isotropic bounding box
  L                  // length of a voxel
  K^max_vox/batch    // max. voxels per batch

// Approx. voxel count and calc. min. Morton index (GPU)
{#V_i}, {M_i} ← calcCountAndMorton({T_i}, Vbox, L)

// Sort triangles acc. to their min. Morton index (CPU/GPU)
if memory usage of sorting < free GPU memory then
  {M_i} ← sortOnGPU({M_i})
else
  {M_i} ← sortOnCPU({M_i})

// Determine triangle batches (CPU)
  j ← 0              // count of batches
  sum ← 0            // voxel sum of current batch
  t ← 0              // triangle count of current batch
  {vo_i} ← {0}       // voxel offsets per triangle
foreach T_i do
    if sum + #V_i < K^max_vox/batch then
      vo_i ← sum
      sum ← sum + #V_i
      t ← t + 1
    else
      // Save Morton stop index, triangle count and voxel count
      batch_j ← createBatch(M_i+1, t, sum)
      j ← j + 1
      sum ← #V_i
      t ← 1
      vo_i ← 0
  batch_j ← createBatch(2^63, t, sum) //last batch
```

Sec. 3.4.5.1), which has to maintain all voxel-attribute pairs for non finalized voxels, so that a memory overflow becomes more probable. Thus, a second user-defined value $K^{max}_{vox/batch}$ is introduced as bound for the maximum number of voxels which can by generated in a batch. Since the maximum voxels $N^{max}_{vox/tri}(T_i)$ per triangle T_i in the batch are known, Eq. 3.8 allows to determine an upper bound for the voxel count per batch. The current batch $B_j = \{T_{j_{min}}, \ldots, T_{j_{max-1}}\}$

is finished and a new batch B_{j+1} is created, if Eq. 3.8 is not fulfilled by T_i. The batch determination is done on CPU by sequentially processing the sorted list of triangles. Respective implementation details are provided in Alg. 3.2.

$$\sum_{i=j_{min}}^{j_{max}} N_{vox/tri}^{max}(T_i) \leqslant K_{vox/batch}^{max} \tag{3.8}$$

3.4.4.3 *Voxelization and Attribute Creation*

Conservative voxelization of [SS10] is used

The triangles are voxelized after the determination of triangle batches. For voxelization, the conservative approach of [SS10] is applied. The summation in Eq. 3.8 allows to define separate memory ranges in a per-triangle voxel memory, so that voxels of each triangle can be created independently. After the voxelization of the triangle batch, a parallel stream compaction removes empty slots of the per-triangle voxel memory. Additionally, the list of valid voxel-attribute pairs, i.e. the *voxel attribute set*, is copied into the Morton queue. This procedure prevents the usage of atomic operations which is usually needed for grid-based parallel voxelization.

Attribute creation is done by texture lookups

The attribute creation is taking place at the same time as the voxelization. While the creation of attributes can be done by arbitrary properties which can be attached to triangles, triangle normals and optionally color information are processed and stored in the proposed implementation. Textures hold the color information. The (u, v)-coordinates for the texture lookup are determined by the barycentric coordinates of the projected voxel center. The (u, v)-coordinates are clamped to the nearest edge of the triangle, if the projection of the voxel center does not lie in the triangle. The texture lookup is performed afterwards. Even dependent texture lookups can be performed at this stage. More visual attributes can be retrieved and used for the current voxel. Thus, the creation of individual voxel attributes can be much more sophisticated than an attribute creation on a per-triangle-basis, as proposed by [BLD14].

3.4.5 *Voxel Stream Processing*

3.4.5.1 *Morton Queueing*

Purpose of the Morton queue

The order for creating the SVT nodes is determined by the Morton indices of the voxel grid. It can not be ensured that all generated voxels of a triangle are usable for SVT creation, because the range of Morton indices covered by a triangle may be large. Thus, the SVT creation of these voxel-attribute pairs is done in a later iteration. As voxel-attribute pairs can not be skipped, the Morton queue is used for maintaining all voxel-attribute pairs from previous batches which

could not be processed yet. Therefore, the Morton queue has a varying size according to the number of remaining voxel-attribute pairs in each iteration.

Processable voxels are extracted

After adding the voxel-attribute set of the current batch to the Morton queue, the resulting set of voxel-attribute pairs is sorted. By splitting the Morton queue at the *stop index* (see Sec. 3.4.4.2 and Fig. 3.13) and an extraction of all voxels with a smaller Morton index, the new voxel-attribute set for SVT creation is determined. This method ensures that no further triangle will intersect any voxel of the extracted voxel-attribute set, i.e. newly created Morton indices will be greater than or equal to the stop index.

Risk of a memory overflow in extreme cases

The number of elements in the Morton queue should be kept small, because remaining voxel-attribute pairs block memory. The triangle subdivision of Sec. 3.4.4.1 tries to limit the range of Morton indices of a triangle to minimize the number of remaining voxels in the Morton queue, which need to be stored during the processing of many batches. However, the prevention of an overflow is not guaranteed in extreme cases, e.g. if a huge number of triangles intersects a single voxel or many triangles span very large Morton ranges.

3.4.5.2 *SVT Creation*

Subtrees are created in parallel over the hierarchy levels

After extracting the voxel-attribute set, the nodes of the SVT can be created bottom-up, because individual Morton indices of upper nodes can be determined by Morton indices of the leaf voxels. To obtain the Morton index of a parent node, the Morton index of its child is divided by the child count of a node, i.e. N^3. The parent indices are determined on each hierarchy level for all nodes to compact them to unique identifiers. Afterwards, these identifiers are used for a parallel node creation. All child nodes of one unique parent index are processed by one GPU thread. Here, voxel attributes and bitmasks are calculated (cf. with "valid masks" in [LK10] and Fig. 3.3). In the bitmask, a non-empty child node is represented by 1, while an empty child node is represented by 0. Each thread executes a modulo division by N^3 for all Morton indices of its child nodes. To obtain the correct position of a node in the bitmask, the result of this division is used for the shift of a single bit which is combined with the final bitmask by a logical OR. Since the sorting order needs to be preserved over the hierarchy levels, the threads are scheduled by the sorted Morton indices of the processable nodes. [ZGHG11] and [SS10] use this method of a breadth-first bottom-up creation of the tree as well. Alg. 3.3 provides further details on the implementation.

Stitch queue buffer for correct attribute determination of parents

To provide arbitrary, even order-dependent or multi-pass operations in the attribute computation, an SVT node can only be finalized, if all of its child nodes have been finalized as well. Therefore, a *stitch queue buffer* is used to store SVT nodes which are required for processing the last immediate parent node on the current hierarchy level.

Algorithm 3.3 : Bottom-up creation of SVTs. Source: [PK15]

```
// Input
  VS        // voxel attribute set
  {M_VS}    // Morton indices of voxel attribute set
  SQ        // stitch queue buffer
  bits      // final bitmasks
  atts      // final attributes

// Process the active voxel attribute set
if VS ≠ ∅ then
    // Create unique attributes for all leaf voxels (GPU), i.e.
    // 1) Calc. offsets to access all attributes of one voxel
    // 2) Create unique voxel attributes
    // 3) Save encoded attributes to CPU
    o_v ← getUniqueVoxels(VS) // reduction & exclusive scan
    VS ← createAttributes(VS.size, o_v, VS)
    atts[leaflevel].appendOnCPU(encodeAttributes(VS))

    // Create SVT subtree levels bottom-up
    foreach l do
        // Allow post-order attribute creation (GPU), i.e.
        // 1) Add stitch queue of level to voxel attribute set
        // 2) Calc. max. parent node index of voxel attribute set
        // 3) Move voxels with max. parent index to stitch queue
          VS.addToFront(SQ[l])    // VS remains sorted
          {M_P} ← {M_VS}/N^3      // Morton indices of parents
          SQ[l] ← VS.extractFromBack(max({M_P}))

        if VS = ∅ then
            // Terminate SVT creation of current batch
            break

        // Create SVT structure (GPU), i.e.
        // 1) Calc. offsets to access all attributes of one node
        // 2) Create unique node attributes and SVT-bitmasks
        // 3) Prepare voxel attribute set for next level
        // 4) Save bitmasks and encoded attributes to CPU
        o_n ← getUniqueParents({M_P}) // reduct. & excl. scan
        a_n, b_n ← createAttribsAndBitmasks(VS.size, o_n, VS)
        VS ← toUpperLevel({M_P}, a_n)
        bits[l].appendOnCPU(b_n)
        atts[l].appendOnCPU(encodeAttributes(a_n))
```

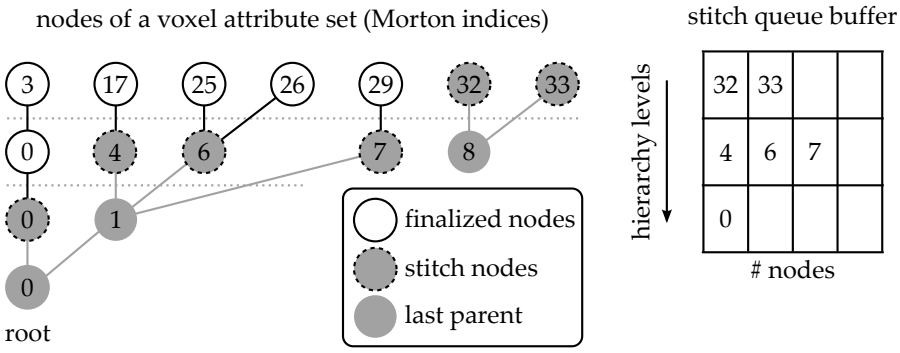


Figure 3.15: The stitch queue buffer maintains intermediate SVT nodes in a queue on each hierarchy level between the processing of triangle batches to allow a post-order attribute creation. An example for processing nodes of a quadtree for a given voxel attribute set is shown. Image source: [PK15]

Those nodes may need to be "stitched" with nodes of the next batches to generate the parent node. An example for the processing of a voxel-attribute set is shown in Fig. 3.15. Here, a quadtree is used for simplicity. An integer division of 4 (quadtree nodes) gives the parent indices of the bottom level. The last parent node 8 contains the child nodes 32 and 33. These two nodes need to be stored in the stitch queue buffer, because future triangle batches may create further child nodes for parent node 8, e.g node 34. The other child nodes 3, 17, 25, 26 and 29 are processed and pushed to the next higher level. The parent indices are calculated again. Since nodes 4, 6 and 7 belong to the largest parent index 1, they are moved to the stitch queue buffer.

Child nodes are maintained between the batches

In the next iteration, the sorted stitch queue of the respective hierarchy level is copied to the front of the current voxel-attribute set, removed from the stitch queue buffer, and processed like all other nodes (see "post-order attribute creation" of Alg. 3.3). The stitch queue buffer has a fixed size and contains a maximum of N^3 nodes per level, because only intermediate SVT nodes are stored. The voxel-attribute pairs, which belong to the last leaf node (stop index, see Fig. 3.13), remain in the Morton queue, while the attributes of the preceding leaf nodes can be generated by the complete list of associated voxel-attribute pairs.

Difference between "SVO Builder queues" and stitch queue buffer

Although the "SVO Builder queues" of [BLD13] and the stitch queue buffer have the same size for N = 2, the functionality is different. The SVO Builder queues are used to create the whole SVO and represent a consistent tree over the levels, while the stitch queues are only used for maintaining unprocessable nodes between the iterations. Furthermore, the contained nodes of the stitch queues do not have a logical connection over the hierarchy levels.

A final step on CPU creates remaining attributes

The stitch queue buffer holds the information of all remaining nodes on each hierarchy level after the last batch is processed. Since

a parallel processing of a few nodes per level does not make much sense at this stage, the SVT creation is finished on CPU (see Alg. 3.4). Therefore, all nodes per hierarchy level are processed sequentially and the information of new nodes is copied to the next coarser level of the stitch queue buffer.

Algorithm 3.4 : Finishing SVT creation & output. Source: [PK15]

```
// Input
  SQ                  // stitch queue buffer
  bits, inds, atts    // final bitmasks, final indices, final attributes

// Finish SVT creation (CPU, see "Create SVT structure"
(Alg. 3.3) bits.append(createNodes(SQ))
atts.append(createNodes(SQ))

// Create indices which point to first child of node (CPU/GPU)
inds ← countBits(bits)
if memory usage of bitmask data < free GPU memory then
  inds ← exclusiveScanOnGPU(inds)
else
  inds ← exclusiveScanOnCPU(inds)

// Save voxel attribute, bitmask and index arrays to disk (CPU)
saveToDisk(atts, bits, inds)
```

3.4.6 *Results*

Overview

The following section presents results of the proposed voxelization method. To compare the approach with other techniques, Sec. 3.4.6.1 evaluates the SVT with N = 2, because it corresponds to an SVO structure, which can be created with other methods as well. Sec. 3.4.6.2 focuses on the influence of N to evaluate the generalization concept itself (see Sec. 3.3.2).

3.4.6.1 *Comparison with related methods*

Comparison with other approaches and evaluation of batch parameters

The performance and the quality of the created attributes are compared with the work of [BLD14][1] and [LK11][2], who thankfully made the source code publicly available for evaluation. Furthermore, the influence of the user-defined parameters for restricting voxels per triangle ($K^{max}_{vox/tri}$) and per batch ($K^{max}_{vox/batch}$) are discussed. An Intel Xeon

1 https://github.com/Forceflow/ooc_svo_builder
2 https://code.google.com/p/efficient-sparse-voxel-octrees/

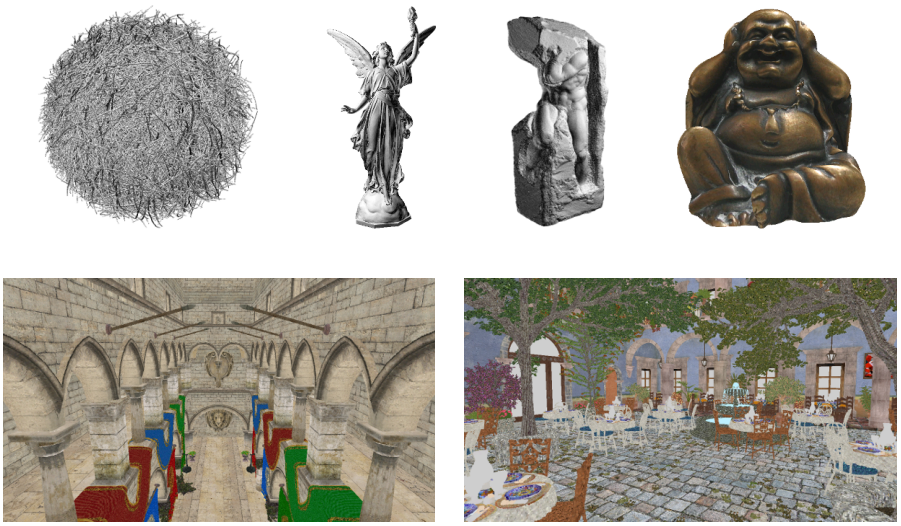


Figure 3.16: The test scenes are shown. Top row from left to right: Hairball, Lucy, Atlas, Buddha; Bottom row from left to right: Sponza, San Miguel. Image source: [PK15]

E5-2623 (3.0GHz) with 64 GB of RAM and an NVIDIA GeForce GTX TITAN Black with 6 GB of memory have been used for evaluation.

Description of test scenes

PERFORMANCE The performance is compared for six test scenes at resolutions of 2048^3, 4096^3 and 8192^3 isotropic voxels. The test scenes Hairball, Lucy and Atlas vary in scene complexities and triangle counts, while Buddha, Sponza and San Miguel, allow a direct comparison between the creation of voxel scenes with and without color attributes (see Fig. 3.16).

Adjustments and requirements for a fair comparison

A few of the default parameters of the compared frameworks are adjusted to provide a fair comparison. For [LK11] the maximum number of available CPU threads is used and voxel scenes without contour information are created. For [BLD14], the maximum memory is increased to the available memory, but the approach does not benefit from more memory, if the next larger subgrid cannot be fully represented, e.g. a memory limit of 8 GB leads to the same partitioning as a memory limit of 63 GB because 64 GB would be needed for representing the grid of the next octree level.

Batch parameters are constant in this comparison

The user parameter "voxels per triangle" $K^{max}_{vox/tri}$ is set to 100 and "voxels per batch" $K^{max}_{vox/batch}$ is set to 10 M (see Sec. 3.4.4) for the performance evaluation. After the comparison with other approaches, the influences of these two parameters are discussed.

Performance results are shown in Tab. 3.2

Tab. 3.2 shows the result of the performance comparison. The whole process with hard disk I/O is included in the timings for [BLD14] and the proposed approach. The time of loading a triangle mesh from hard disk to RAM is omitted for [LK11] due to restrictions of the available file-import. It would have been more unfair to add the import timings instead of omitting the timings because the load-

Scene		Hairball (2.8 M triangles)			Lucy (28.0 M triangles)			Atlas (506.5 M triangles)		
Resolution		2048	4096	8192	2048	4096	8192	2048	4096	8192
[LK11]		274.4 s	763.7 s	2657.8 s	964.3 s	1001.9 s	1097.4 s	-	-	-
[BLD14]		134.4 s	759.2 s	4459.9 s	17.5 s	40.7 s	97.9 s	223.3 s	351.4 s	676.3 s
Proposed algorithm		83.0 s	281.9 s	1195.5 s	11.7 s	16.9 s	30.3 s	270.0 s*	239.8 s*	345.7 s*
Scene		Buddha (30.3 K triangles)			Sponza (262.3 K triangles)			San Miguel (10.1 M triangles)		
Resolution		2048	4096	8192	2048	4096	8192	2048	4096	8192
w/o col.	[LK11]	14.3 s	59.9 s	243.2 s	24.8 s	92.1 s	364.2 s	(140.6 s)	(153.39 s)	(165.9 s)
	[BLD14]	15.4 s	66.6 s	372.1 s	23.1 s	83.6 s	437.1 s	9.5 s	26.5 s	107.4 s
	Proposed algorithm	4.4 s	14.0 s	49.4 s	6.8 s	24.1 s	97.9 s	6.2 s	12.0 s	32.6 s
w col.	[LK11]	16.5 s	65.3 s	262.6 s	31.3 s	112.4 s	428.7 s	(171.7 s)	(187.5 s)	(203.9 s)
	[BLD14]	55.4 s	166.3 s	611.6 s	52.1 s	363.7 s	1416.9 s	13.0 s	37.6 s	228.3 s
	Proposed algorithm	5.1 s	17.9 s	50.4 s	11.0 s	30.8 s	111.6 s	13.2 s	20.4 s	44.3 s

Table 3.2: Timing comparison for different scenes (depicted in Fig. 3.16) at different resolutions: The three scenes in the upper row have strongly varying triangle counts, mesh attributes and scene complexity. The three scenes in the bottom row allow a direct comparison between a voxelization with and without color creation. *: Average over three runs; (...): Scene could be voxelized, but not rendered. Source: [PK15]

Resolution		2048	4096	8192	2048	4096	8192	2048	4096	8192
Scene		Hairball (2.8 M triangles)			Lucy (28.0 M triangles)			Atlas (506.5 M triangles)		
Pre-processing		4.6 s	12.8 s	43.5 s	5.4 s	6.0 s	6.6 s	198.4 s*	153.2 s*	213.0 s*
Voxelization		54.2 s	179.4 s	674.0 s	3.8 s	6.7 s	13.4 s	44.4 s*	57.6 s*	86.9 s*
SVT Creation		12.6 s	53.4 s	257.5 s	0.5 s	1.5 s	4.5 s	4.9 s*	5.8 s*	14.1 s*
Tri.Mem. bef. Subdiv.		97.9 MB			962.3 MB			17.0 GB		
Tri.Mem. aft. Subdiv.		639.4 MB	1.5 GB	4.3 GB	962.5 MB	963.1 MB	965.8 MB	17.0 GB		
SVT Mem. Consumption		1.1 GB	4.7 GB	19.2 GB	41.1 MB	164.6 MB	658.8 MB	55.17 MB	222.0 MB	891.0 MB
Scene		Buddha (30.3 K triangles)			Sponza (262.3 K triangles)			San Miguel (10.1 M triangles)		
w/o col.	Pre-processing	0.8 s	0.9 s	1.6 s	0.9 s	1.4 s	3.2 s	2.4 s	2.5 s	3.1 s
	Voxelization	1.7 s	6.0 s	22.8 s	3.2 s	10.8 s	42.1 s	2.5 s	5.5 s	16.9 s
	SVT Creation	0.7 s	2.9 s	11.8 s	1.1 s	4.5 s	20.3 s	0.4 s	1.6 s	5.9 s
	Tri.Mem. b. Subdiv.	1.0 MB			9.0 MB			347.5 MB		
	Tri.Mem. a. Subdiv.	16.3 MB	55.6 MB	203.9 MB	49.4 MB	145.5 MB	510.1 MB	355.3 MB	380.6 MB	494.1 MB
w col.	Pre-processing	0.9 s	1.5 s	2.0 s	1.4 s	2.0 s	4.8 s	4.0 s	4.4 s	4.7 s
	Voxelization	1.9 s	6.3 s	24.5 s	3.6 s	12.0 s	47.0 s	2.8 s	6.8 s	19.3 s
	SVT Creation	0.9 s	3.2 s	11.8 s	1.4 s	4.5 s	22.6 s	0.6 s	2.3 s	7.9 s
	Tri.Mem. b. Subdiv.	1.8 MB			15.5 MB			598.4 MB		
	Tri.Mem. a. Subdiv.	28.0 MB	95.7 MB	351.1 MB	85.1 MB	250.6 MB	878.4 MB	611.8 MB	655.5 MB	851.0 MB
SVT Mem. Consumption		137.2 MB	549.1 MB	2.15 GB	259.9 MB	1.0 GB	4.1 GB	68.4 MB	272.7 MB	1.1 GB

Table 3.3: Memory consumption (triangles before subdivision, triangles after subdivision and final SVT) and performance breakdown for the test scenes are shown.; *: Average over three runs; Please note that the memory consumption of a bitmask is not aligned to 32 bits like in Tab. 3.5, because N is fixed to 2 in this evaluation. Source: [PK15]

ing of all scenes (except the Atlas scene) in a binary format needed less than 2 seconds.

Additional information is provided in Tab. 3.3

Additional information regarding memory requirements and a performance breakdown of the proposed approach can be found in Tab. 3.3. The memory usage of the input triangle soup before and after subdivision as well as memory usage of the resulting SVT with compressed attributes is provided. The triangle footprint without color creation is 36 byte, while it is 62 byte with color creation. To store the test scenes in a full grid representing the used compression with 16-bit normal and 16-bit color data per voxel, 32 GB, 256 GB and 2 TB of memory would be needed for the resolutions 2048, 4096 and 8192, respectively. Therefore, the SVT topology drastically reduces the memory requirement already, while the used compression of the attributes reduces the memory usage further. In the given implementation, the SVT memory usage between scenes with and without color creation does not differ, because 16 bit for color are always allocated even if color creation is turned off. Furthermore, the performance is given for individual parts of the proposed method. The three main parts are pre-processing, voxelization and SVT creation.

Sparsity has a much larger influence on performance than the triangle count

Despite the comparably low triangle count, the Hairball scene at a resolution of 8192 leads to the longest processing time. It follows that the sparsity or scene complexity has a much larger influence on performance than the number of triangles, because more voxels contribute to the final tree. In comparison with [LK11], the Hairball at resolution 8192 is processed in roughly half the time while the proposed approach is nearly four times faster than [BLD14].

High regularity of the input mesh leads to a good GPU load-balancing

For [LK11], the triangle availability on all levels gets disadvantageous in the Lucy scene. The scene is voxelized between one and two orders of magnitude faster with [BLD14] and the proposed method, because only leaf nodes use the triangles directly. Furthermore, the SVT approach voxelizes the scene faster than all other scenes at a resolution of 8192, because the high regularity of the input mesh leads to a good load-balancing for GPU processing.

Sorting a massive number of triangles is slower than triangle partitioning

The Atlas dataset has the largest triangle count of the test scenes. The scene could not be processed with the framework of [LK11]. This dataset is the only one for which the proposed triangle sorting cannot be done on GPU, i.e. CPU sorting was applied here. A large variation in the timings was encountered, thus average timings over three runs for each resolution are given. The minimum timings for the resolutions 2048, 4096 and 8192 are 254.8 s, 231.7 s and 300.5 s, respectively. Furthermore, the triangle sorting for a resolution of 2048 was around 40 seconds slower than for a resolution of 4096. It is assumed that the used radix sort needs larger buckets due to less Morton indices for the same triangle count in the limited available memory which leads to memory fragmentation. The only case where [BLD14] performs faster is the voxelization of Atlas at a resolution of 2048. Here, the

triangle partitioning to a voxel grid with low resolution is beneficial in comparison to a sorting of a large number of triangles.

GPU is more beneficial if hardware-supported functions are used

The comparison between a voxelization with color and a voxelization without color of the scenes Buddha, Sponza and San Miguel shows a performance decrease by a factor of two to three for [BLD14]. The SVT voxelization and the method from [LK11] show a smaller performance drop between colored and uncolored scene, but the proposed technique is three to fives times faster than [LK11]. It seems that the SVT method has a performance drop by a factor of two for Sponza and San Miguel at a resolution of 2048, but here only the time for texture loading as a constant value adds to the voxelization time which is roughly the same in this case. It can be seen that this effect is alleviated for higher voxel resolutions of the scenes.

Scene sparsity has the largest influence on performance

It follows again, that the sparsity of the scene has a larger influence on the timing than the triangle count, if the timings for Sponza and San Miguel are compared. Although San Miguel has roughly 40 times more triangles, it creates a sparser scene due to large triangles surrounding the building. On the finest level at a resolution of 8192, around 164 M voxels were created for San Miguel and 636 M voxels were created for the Sponza. Furthermore, the largest performance gain in comparison to [BLD14] is shown for Sponza with color attributes at 8192. Here, the color creation and the sparsity of the scene lead to the performance drop of [BLD14]. The voxelization of San Miguel failed with the approach of [LK11]. Although no errors have been thrown during the processing, the rendering did not show any output, i.e. the result is corrupted.

Similar quality to rasterization or SVO creation of [LK11]

ATTRIBUTE CREATION The voxelization of an extracted plant model from the San Miguel scene with an alpha channel in its texture is used to show the different visual results regarding attribute creation (see Fig. 3.17). The approach of [BLD14] (Fig. 3.17 (d)) clearly lacks visual quality, because all voxels of a triangle are set to the same average triangle color. Fig. 3.17 (c) and (f) show the best and the worst result achieved by [LK11], respectively. The voxelization with contour and postfiltering is shown in Fig. 3.17 (c), while the voxelization without contour information and without image-based post-processing is shown in Fig. 3.17 (f). It can be seen, that the texture attributes are considered and a sophisticated attribute creation is possible. For comparison, a simple (texture) color averaging (see Fig. 3.17 (e)) is used in the SVT creation. While both methods have the same properties for attribute creation on the leaf level, the main difference is the access to all triangles on coarser hierarchy levels of [LK11]. This accessibility is not useful for out-of-core methods, because the root node would need to access all triangles.

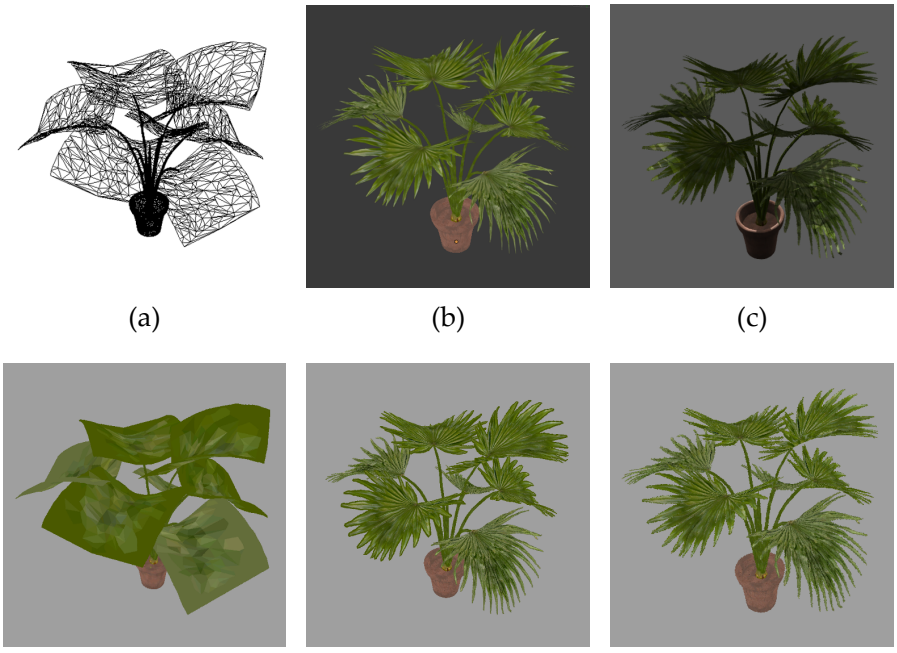


Figure 3.17: Comparison between the attribute creation methods that shows wireframe (a), rasterization (b), contour information per voxel, illumination and image filtering of [LK11] (c), attribute creation of [BLD14] (d), attribute creation of SVT (e) and unfiltered voxel structure of [LK11] (f). Image source: [PK15]

BATCH PARAMETERS The influence of the two user-defined parameters "voxels per triangle" $K^{max}_{vox/tri}$ and "voxels per batch" $K^{max}_{vox/batch}$ (see Sec. 3.4.4.1 and Sec. 3.4.4.2, respectively) is examined in the following. The Hairball at resolution of 2048 and the Sponza with color creation at a resolution of 4096 serve as test scenes for evaluating the parameter influence with respect to memory usage, triangle statistics and timing. For evaluation purposes, the values are set to worst cases. Since the out-of-core property gets unreliable for these extremes, they should not be used in practical usage. Only an empirically determined range of $K^{max}_{vox/tri}$ and $K^{max}_{vox/batch}$ provides the out-of-core property.

Influence of "voxels per triangle" and "voxels per batch"

The influence of $K^{max}_{vox/tri}$ and $K^{max}_{vox/batch}$ to memory usage per batch is shown in Fig. 3.18. Minimum, median and maximum memory size of the batches are shown for each test. Since $K^{max}_{vox/batch}$ acts as a global voxel counter, it can be seen that $K^{max}_{vox/tri}$ has less influence than $K^{max}_{vox/batch}$. If $K^{max}_{vox/tri}$ gets smaller, more triangles need to be created and stored in memory for voxelization. It follows, that the memory usage gets higher for all configurations with a limit to 10 for $K^{max}_{vox/tri}$. A nearly constant behavior can be deduced, if $K^{max}_{vox/tri}$ is increased. A memory overflow occurs for $K^{max}_{vox/batch} = 50$ M voxel and $K^{max}_{vox/tri} = 10$ in the Sponza scene. This parameter combination leads to many small triangles that need to be maintained per batch so

Influence on memory usage per batch

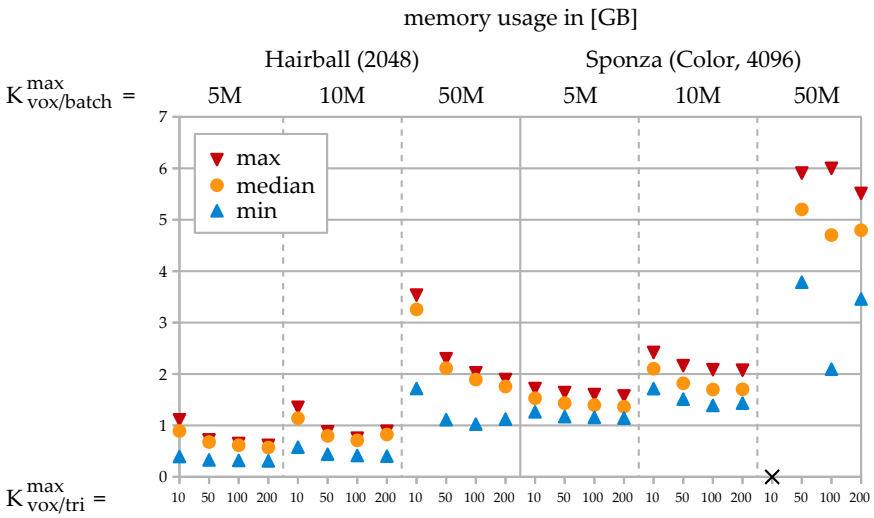


Figure 3.18: Influence of the parameters $K^{max}_{vox/tri}$ with [10, 50, 100, 200] and $K^{max}_{vox/batch}$ with [5M, 10M, 50M] to the GPU memory usage in GB. Image source: [PK15]

that the memory requirements are increased. The 4 Sponza measurements for $K^{max}_{vox/batch} = 50$ M voxel do not seem to follow a pattern like the other configurations. The strong variation of the memory usage between the batches can be explained by large Morton index ranges which result from a large batch size with many triangles. Here, the locality of created voxels in terms of Morton indices is harder to ensure, so that more voxels remain in the Morton queue until they are processable for SVT creation.

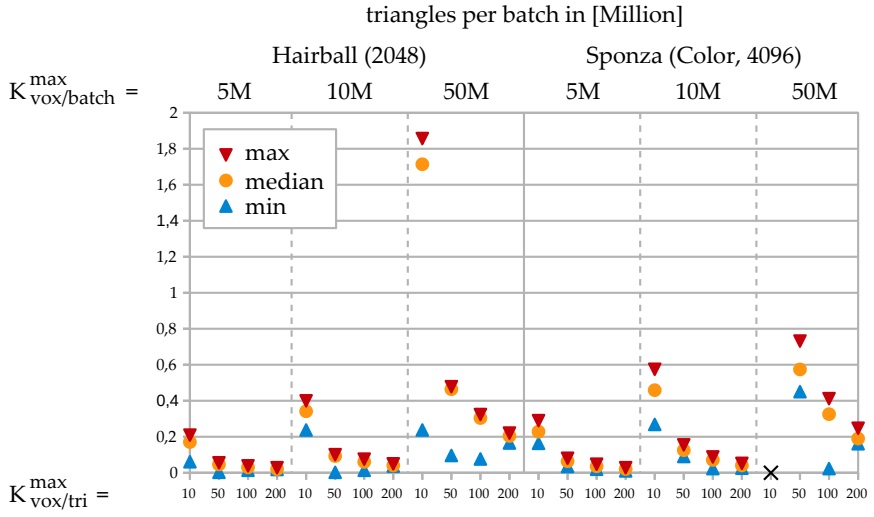


Figure 3.19: Influence of the parameters $K^{max}_{vox/tri}$ with [10, 50, 100, 200] and $K^{max}_{vox/batch}$ with [5M, 10M, 50M] to the triangles per batch in Million. Image source: [PK15]

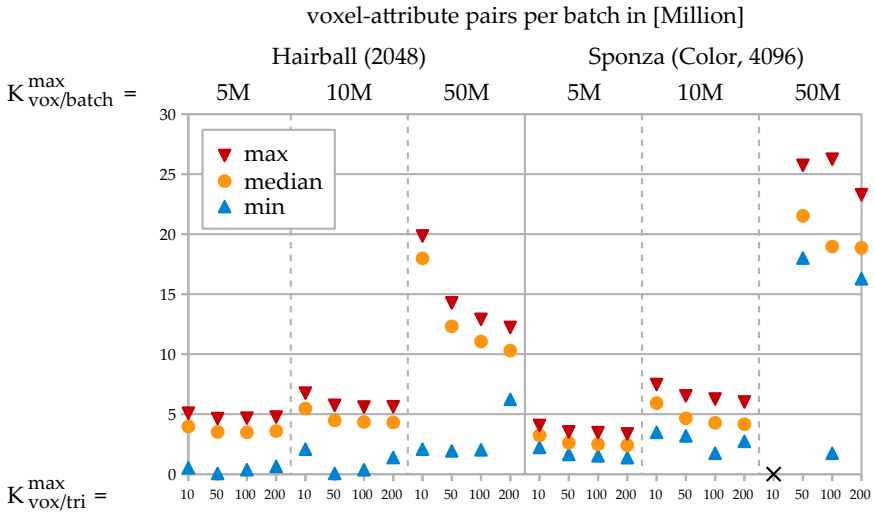


Figure 3.20: Influence of the parameters $K^{max}_{vox/tri}$ with [10, 50, 100, 200] and $K^{max}_{vox/batch}$ with [5M, 10M, 50M] to the number of created voxel-attribute pairs per batch in Million. Image source: [PK15]

Influence on triangles and voxel-attribute pairs per batch

Fig. 3.19 shows triangles per batch and Fig. 3.20 shows the number of created voxel-attribute pairs. If $K^{max}_{vox/batch}$ is increased or $K^{max}_{vox/tri}$ is decreased, more triangles are consumed in a batch. Therefore, more voxel-attribute pairs are created. The differing triangle shapes explain the smaller variation of median and maximum voxel-attribute pairs for the Hairball in comparison to the Sponza. While the Hairball consists of similar triangle shapes, the Sponza has more differing triangle shapes which lead to more varying voxel counts per triangle.

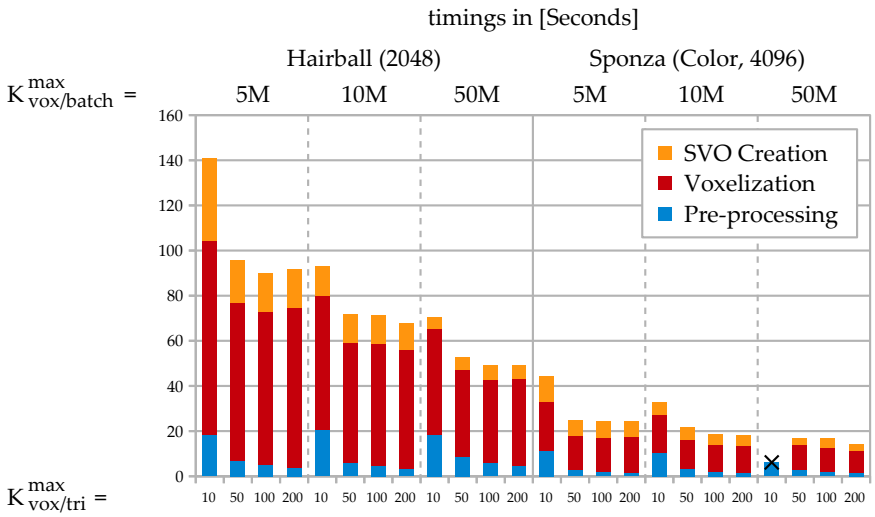


Figure 3.21: Influence of the parameters $K^{max}_{vox/tri}$ and $K^{max}_{vox/batch}$ to the timings in seconds for the steps pre-processing, voxelization and SVO creation. "x": memory overflow. Image source: [PK15]

	10	50	100	200
Hairball (2.8 M)	90.4 M	28.9 M	18.6 M	12.1 M
Sponza (262.3 K)	35.2 M	7.9 M	4.2 M	2.4 M

Table 3.4: Resulting triangle counts after subdivision for different values of $K^{max}_{vox/tri}$. Source: [PK15]

Influence on the performance

Fig. 3.21 shows the behavior of the performance if $K^{max}_{vox/tri}$ and $K^{max}_{vox/batch}$ are varied. Additionally, Tab. 3.4 provides the generated number of triangles of $K^{max}_{vox/tri}$ after the subdivision has been performed. It allows to evaluate the correlation between the correct number of processed triangles and the timings. The performance of the triangle pre-processing solely depends on $K^{max}_{vox/tri}$. Here, the performance drops if $K^{max}_{vox/tri}$ decreases, because the triangles need a stronger subdivision and more triangles need to be processed. If $K^{max}_{vox/batch}$ gets higher, the performance of the voxelization improves, because more triangles can be processed in parallel and the number of batches is reduced. $K^{max}_{vox/tri}$ does not seem to influence the voxelization performance much. An exception is the value of 10 which leads to a much higher triangle count (cf. Tab. 3.4) and a performance drop. The performance of SVO creation is mainly influenced by $K^{max}_{vox/batch}$, because the tree construction consumes voxels instead of triangles. The scene processing slows down for the Hairball with $K^{max}_{vox/batch} = 5$ M and $K^{max}_{vox/tri} = 10$. In this worst case, 3 times more triangles are created in comparison to a scene processing with $K^{max}_{vox/tri} = 50$ (cf. Table 3.4). Therefore, the triangle count per voxel gets more heterogeneous for the post-order attribute creation, which leads to idle threads that process less triangles per voxel.

3.4.6.2 *Influence of N*

Influence of N is evaluated and further explanations are given

The influence of N on the creation of the SVT is evaluated in respect to memory consumption and performance. This evaluation has been performed on an Intel Core i7-930 (2.80GHz) with 24 GB of RAM and an NVIDIA GeForce GTX 680 with 2 GB of memory. To allow an equal scene resolution for a fair comparison between varying values of N, the bounding cube which contains the scene geometry is scaled appropriately. Therefore, the largest extent of one dimension in the scene is independent from the maximum possible resolution of the respective N and is represented by the desired resolution instead. Five scenes have been used for evaluation. The Hairball is voxelized at the resolutions 250, 500 and 2000, while Sponza with color attributes and Lucy are voxelized at a resolution of 4000 and 8000, respectively.

Memory consumption for voxelized surfaces is optimized with $N = 5$

MEMORY CONSUMPTION The influence of N on the memory consumption is shown in Tab. 3.5. The behavior between the columns

	Hairball			Sponza	Lucy
	250	500	2000	4000	8000
2	14.61	56.8	1225.5	1147.7	718.0
3	10.15	45.1	851.7	783.4	493.7
4	9.32	41.6	785.3	727.4	454.8
5	9.07	40.9	771.9	711.5	448.8
6	8.98	40.52	772.8	716.0	451.3
7	8.91	40.54	777.9	722.5	456.2
8	8.8706	40.6	784.1	733.3	461.5
9	8.8708	40.8	796.5	740.1	471.0
10	8.89	41.0	810.6	755.6	482.0
Att.	7.83	34.0	641.0	577.1	359.1
A%	88.3%	84.0%	83.0%	81.1%	80.0%
S%	86.86%	92.86%	97.9%	99.764%	99.982%

Table 3.5: Influence of N (left column) on memory consumption of the SVT in MB. 5 scenes with increasing sparsity (S%) are given from left to right: Hairball (250, 500, 2000), Sponza (4000 with Color) and Lucy (8000). The memory consumption of the leaf voxel attributes (Att.) and the percentage of total memory consumption (A%) for the most efficient N (blue cells) are shown.

is similar. All scenes have the highest memory consumption for the choice of N = 2. With an increasing N, the consumption decreases. Starting from the optimal N, the consumption increases again. The blue cells show the best choice of N for $N = \{2, .., 10\}$. N = 5 is the best choice for three scenes (Hairball@2K, Sponza@4K and Lucy@8K) which vary in resolution and complexity. The only similarity between these scenes is the representation of thin surfaces, because of the high resolutions.

Scene sparsity influences the choice of the optimal N

For the Hairball at resolutions of 250 and 500 voxels, the surfaces are not resolved and merged in the voxels. They can be interpreted as thick surfaces or a volume representation. This leads to a lower sparsity of the scenes and the choice of the optimal N varies. The bitmasks have a higher number of set voxels and the topology is flattened due to a larger N. For the other scenes with a higher sparsity, large empty bitmasks waste more memory than smaller bricks with more hierarchy levels.

Influence of topology to memory consumption

The attribute row (Att.) in Tab. 3.5 shows the memory consumption of the leaf attributes, while their percentage of the total memory consumption with the optimal N is provided in the row A%. It can be seen that the relative memory consumption of the attributes is decreasing for an increasing sparsity. Therefore, the topology has a

N	64 ptrs., 32 al.	32 ptrs., 32 al.	32 ptrs., 8 al.
2	1420.4	1225.5	1079.4
3	921.9	851.7	851.7
4	821.4	785.3	785.3
5	793.7	771.9	771.9
6	787.4	772.8	769.1
7	788.4	777.9	775.3
8	792.1	784.1	784.1
9	802.7	796.5	796.5
10	815.6	810.6	806.9

Table 3.6: Influence of N on memory consumption of the SVT in MB with different implementation details. The blue cells mark the optimal memory consumption of each adjustment. The Hairball@2K is created with different adjustments in the SVT creation. From left to right: 64-bit child pointers and 32-bit aligned bitmasks, 32-bit child pointers and 32-bit aligned bitmasks, 32-bit child pointers and 8-bit aligned bitmasks

larger percentage of the memory consumption. The reasons are a decreasing fill rate of the bitmasks and a reduced number of leaf voxels.

The optimal choice of N can vary due to implementation details

Furthermore, it can be seen that the differences between the optimal N and the neighboring values of N lead to very similar memory consumptions. Different adjustments like changing the alignment of bitmasks from 32 bits to 8 bits or switching child pointer sizes between 32 bit and 64 bit directly change the optimal N (see blue cells in Tab. 3.6). It follows that the right choice of N can vary depending on implementation details for extending or adjusting the proposed SVT.

Performance of creation is slightly improved by increasing N

PERFORMANCE The influence of N on the performance is shown in Tab. 3.7. N is limited to 5 in this evaluation, because the maximum allowed bitmask size is 128 bit in the current implementation. While the correct memory consumption can be calculated for larger values of N, the bitmasks are not consistent and can not be rendered. It can be seen that the creation of SVTs gets just slightly faster for larger values of N. Two reasons are the reduced number of bricks which are processed in parallel and the reduced data which needs to be stored on disk, but the performance gain does not seem to be significant. The Sponza scene shows that an increased workload for attribute creation might lead to a larger performance gain when more complex operations are performed on GPU.

	Hairball			Sponza	Lucy
	250	500	2000	4000	8000
2	3.36	6.94	85.45	29.37	31.95
3	3.21	6.59	80.9	25.78	29.68
4	3.23	6.54	80.03	25.25	28.9
5	3.21	6.44	78.23	24.07	28.5
R%	95.5%	92.8%	91.6%	82.0%	89.2%

Table 3.7: Influence of N (left column) on performance for creating an SVT in seconds. The blue cells mark the most performant SVT creation for each scene. The ratio (R%) between slowest and fastest SVT creation is given in the last row. 5 scenes are shown from left to right: Hairball (250, 500, 2000), Sponza (4000 with Color) and Lucy (8000).

3.5 RENDERING

3.5.1 *Overview*

Surface rendering of SVOs is extended

Unlike the rasterization of polygonal meshes, voxel rendering is no standardized technique on current GPUs. Instead, an efficient rendering of voxels has to be implemented explicitly. Commonly, raycasting is used as the fundamental method to allow a fast traversal of the voxel structure. Depending on individual properties of a voxel representation, the traversal needs to be adjusted and optimized. The proposed ray traversal generalizes an efficient surface rendering of SVOs to a hybrid rendering of surfaces and sparse volumes of N^3-trees in the form of SVTs with $N \in \{2, \dots, 5\}$. The focus lies on thick surfaces, which can not be represented with surface approaches and which would lead to large memory overheads if represented by true volumetric representations, because they are still sparse.

Structure of the following section

This section discusses the implemented ray traversal which allows to process the proposed SVT. First, an overview of related work for rendering sparse voxel data and an algorithm overview is provided. Afterwards, a description of the proposed ray traversal and a comparison with the adapted approach of [LK10] is given. Next, further optimizations in the implementation are shown. A discussion on results and limitations closes the section.

3.5.2 *Related Work*

Voxels are used for volume and surface rendering

Raycasting of voxel structures for volume and surface rendering is a fundamental and large research topic in CG. The main purpose of the proposed SVT is a surface rendering with thickness properties in sparse scenes. Hence, a discussion on methods for direct volume

rendering of full volumes is not part of the following section. Recent overviews of volume rendering can be found in [BRGIG*14] and [BHP15].

Traversal of voxel structures is based on rasterization

The rendering of surfaces with voxels is based on fundamental rasterization techniques like the digital differential analyzer (DDA) or the Bresenham algorithm [Bre65]. In general, a line or a ray is traversed along its dominant axis direction to determine the intersected voxels of a uniform grid. Depending on the slope, it is calculated if a step to a voxel in the other axis directions is necessary before continuing the traversal in the main axis direction.

[AW87] provides fundamental approach for voxel grid traversal

In respect of a voxel grid traversal, those rasterization techniques do not allow to determine all voxels which are really traversed by the ray. Therefore, [FTI86] and [AW87] adjusted the DDA to remove this limitation. While [FTI86] still needs steps in the grid without calculating the correct succeeding voxel beforehand, [AW87] removed this necessity and created a fast and reliable approach which serves as fundamental technique for obtaining a surface rendering from a ray traversal of a voxel grid. The approach of [AW87] is incorporated in the proposed SVT rendering. Hence, a more detailed discussion is provided in Sec. 3.5.4.

[LK10] uses a combination of [AW87] and a hierarchical traversal

[LK10] presents a very performant rendering of SVOs by combining the approach of [AW87] with a stack-based hierarchical traversal. Every time the ray switches between hierarchy levels, a stack stores the current state. Furthermore, the voxels of [LK10] do not contain references to triangles anymore. Instead, a rendering of contours or common voxels is implemented (see Sec. 3.3.1 for further details of the provided data structure). The proposed SVT of this thesis uses the data structure and voxel rendering of [LK10] and extends it to a rendering of N^3-trees by using [AW87] as well. Therefore, the hierarchical traversal and differences to the rendering of [LK10] are discussed in Sec. 3.5.4. Such a stack-based hierarchical traversal is used in succeeding voxel rendering approaches as well. Specially, [KSA13] and [DKB*16] use it to traverse improved voxel structures with lower memory requirements (see Sec. 3.3.1).

[CNLE09] uses stackless tree traversal and volume rendering

Another important voxel rendering is presented by [CNLE09]. In comparison to [LK10], the hierarchical voxel tree traversal is implemented without a stack. Instead, the traversal is based on the kd-restart algorithm of [HSHH07]. The traversal always starts from the root and continues to finer levels of the voxel tree until the appropriate level-of-detail is found. Depending on a constant or a varying subvolume in the intersected voxel, an analytical integration or an uniform ray sampling are used, respectively. The exit position of the ray is used for the next descent from the root level. Fig. 3.22 shows this principle.

Other voxel approaches improve on rendering quality and surface details

Other trends in voxel rendering are improved rendering quality and higher surface details without increasing the number of voxels.

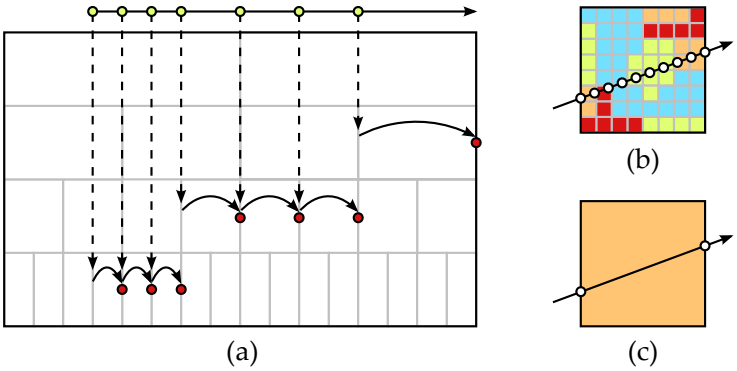


Figure 3.22: The tree traversal of [CNLE09] with the kd-restart of [HSHH07] is shown. In the example, a one-dimensional tree hierarchy is traversed by descending at different ray positions (a). If the ray intersects a voxel that corresponds to the necessary level-of-detail, the subvolume is either traversed by an usual uniform ray casting (b) or an analytical integration between start and end position of the ray (c) depending on the referenced information. The end position (red dots in (a)) of the ray serves as the new position (green dots in (a)) for the next descent from the root node.

Here distance fields ([DW15]) or noise functions ([CNLE09]) are used. In respect to a THz simulation, the approach of [HN12] is interesting, because it uses statistical roughness parameters, which are the basis for electromagnetic wave simulations as well. Those statistical descriptions inside the voxels have the goal to add details to a larger macroscopic volumetric representation. The approach focuses on visible light and cone tracing ([CNS*11]) through hierarchy levels for a mipmapping approach. One drawback of the method is the high memory consumption per voxel.

3.5.3 *Algorithm Overview*

SVT rendering consists of 6 functionalities

The rendering of the proposed SVT is achieved by extending the stack-based traversal of [LK10]. Instead of rendering surfaces of SVOs, the adjusted method allows to render thick surfaces of SVTs with $N = \{2, 3, 4, 5\}$. The general rendering of SVTs consists of 6 basic operations which are used in [LK10] as well:

- Initialize: Required variables for processing are initialized
- Push: A descent to a finer hierarchy level of the SVT is performed
- Step: The next voxel on the current hierarchy level is determined

- Pop: An ascent to a coarser hierarchy level of the SVT is performed
- IntersectionTest: It is tested if the ray intersects the current voxel
- AttributeCalculation: Ray properties are calculated from voxel attributes

Overview of the basic traversal

Alg. 3.5 shows how those basic functionalities are used in the ray traversal. First, the initialization is done (Sec. 3.5.4.1) and the ray is tested for intersection with the bounding box which contains the voxel representation. If the ray missed the voxel grid, the background color is drawn. If the ray is inside, the main loop is entered. It is executed until the ray leaves the voxel grid or terminates due to a stopping criterion. While sampling the ray, traversed voxels are determined and an intersection test between currently traversed voxel and scene representation is performed in every iteration of the main loop (Sec. 3.5.4.2). If an intersected voxel is found, either the attribute of the voxel is processed for adjusting the final pixel color (Sec. 3.5.4.3) or a descent (push) to finer hierarchy levels is performed (Sec. 3.5.4.4). In the case of enabled volume rendering or a ray miss, a step to the next voxel on the current hierarchy level is performed (Sec. 3.5.4.4) and the next iteration starts if the current level is not traversed completely. An ascent (pop) to coarser hierarchy levels is done, if the determined voxel is outside the current brick (Sec. 3.5.4.4).

Algorithm 3.5 : Overview of the traversal algorithm

```
Initialize()
if ray intersects bounding box of voxel grid then
    while ray is inside bounding box do
        IntersectionTest()
        if ray hits a non-empty voxel then
            if stopping criterion fullfilled or finest level reached then
                AttributeCalculation()
                if surface rendering is enabled then
                    break
            else
                Push()
        Step()
        if next voxel is outside of brick then
            Pop()
    writeColor()
else
    writeBackgroundColor()
```

Overview of optimizations

The ray traversal can be optimized in several ways, which are based on proposed techniques of [LK10] as well. To achieve a performance gain, the initial start positions for the rays which are used in the rendering at the final resolution can be optimized by a rendering at a coarser resolution (Sec. 3.5.5.1). If the level-of-detail for the SVT is larger than a pixel of the final rendering can show, the ray traversal can be determined earlier at coarser hierarchy levels to save performance as well (Sec. 3.5.5.2). Another possibility to increase the performance is the use of data types which are used as so-called bitstacks to reduce the accessing time to memory locations which can only be accessed with larger delays (Sec. 3.5.5.3).

3.5.4 *Ray Traversal*

Basic functionalities are discussed in the following subsections

The following subsections describe the basic functionalities of the ray traversal and compare them with the approach of [LK10], which is the basis of the implemented traversal. The basic functionalities consist of initialization, intersection of ray and voxels, attribute determination and hierarchical traversal. The hierarchical traversal consists of functionalities for descending and ascending in the hierarchy levels of the SVT. Furthermore, the stepping inside one hierarchy level is explained. Tab. 3.8 gives an overview on all variables which are used in this section.

3.5.4.1 *Initialization*

The intersection of ray and voxel grid needs to be computed

At the beginning of the traversal, it is necessary to determine if the ray $\vec{r}$ intersects the bounding box bbox which contains the cubical voxel grid. A ray $\vec{r}$ is defined by Eq. 3.9:

$$\vec{r}(t) = \vec{o} + t \cdot \vec{d} \qquad with\, t \geqslant 0 \tag{3.9}$$

The t-span of the ray that intersects bbox is defined by t_{min} and t_{max}, which define the entry and exit point, respectively. bbox is a cube which is defined by 6 axis-aligned planes. Therefore, on each coordinate axis a minimum ($\vec{c}_{min}$) and a maximum value ($\vec{c}_{max}$) are given. To determine all possible intersections of $\vec{r}$ with bbox, the 6 t-values are computed by Eq. 3.10 which is solved for $\vec{t}$ in Eq. 3.11:

$$\vec{c}_{\{min,max\}} = \vec{o} + \vec{t}_{\{min,max\}} \cdot \vec{d} \tag{3.10}$$

$$\vec{t}_{\{min,max\}} = \frac{1}{\vec{d}} \cdot \vec{c}_{\{min,max\}} + \frac{-\vec{o}}{\vec{d}} \tag{3.11}$$

Ray needs a t-span with a valid entry and exit point

The fractions $\frac{1}{\vec{d}}$ and $\frac{-\vec{o}}{\vec{d}}$ are obtained by element-wise division. As stated in [LK10], these fractions can be precomputed in the initialization to check further intersections by one multiplication- and one add-operation. After all 6 intersections are found, Eq. 3.12 and 3.13 are

applied, because only the last entry point and the first exit point define the valid t-span of the intersection. If $t_{max} > 0$ and $t_{min} < t_{max}$, all other parameters are initialized as well, because the ray traversal needs to be executed.

$$t_{min} = max(\vec{t}_{min}) \tag{3.12}$$

$$t_{max} = min(\vec{t}_{max}) \tag{3.13}$$

Parameters for traversal of [AW87] are initialized

Mainly, those parameters consist of stack variables for the voxel traversal of [AW87]. The basic idea of [AW87] is an incremental calculation of all intersections of a ray and subdivision planes of a uniform grid. Fig. 3.24 depicts the approach. A voxel v in the 3D grid is identified by integer values x, y and z. The first intersected voxel $v_{x,y,z}$ is determined by testing if the origin $\vec{o}$ is inside the voxel grid. If $\vec{o}$ is outside, $v_{x,y,z}$ is determined by calculating the smallest t for entering the grid. The sign of the direction $sign(\vec{d})$ determines if x, y and z need to be incremented or decremented for the plane intersections. Hence, constant steps $s_{\{x,y,z\}}$ are initialized with -1 or +1. Furthermore, $\Delta t_{\{x,y,z\}}$ serve as directed stepping distances on the ray between two succeeding subdivision planes in x-, y-, z-direction and $m_{\{x,y,z\}}$ store t for the next intersections with a subdivision plane along x-, y-, z-direction.

Overview on variables provided in Tab. 3.8

Additionally, variables for accessing the content of the SVT are necessary, e.g. attributes and child nodes need to be accessed correctly. Tab. 3.8 shows which variables are stored once, which are stored on every hierarchy level and which are calculated from stack information every time.

Initialization concept

In comparison to [LK10], the concept of the initialization is the same but the implementation is more complex due to the generalization of SVOs to SVTs. While [LK10] exploits specialized optimizations for SVOs which leads to a reduced number of variables and a decreased memory consumption of the stack, the generalized SVT needs to maintain additional information to cover all cases for $N = 2, 3, 4, 5$. A simple example is the storage of steps $\Delta p_{\{x,y,z\}}$ between planes in the brick. While the step is fixed to the half of the brick in the SVO and is used implicitly, the SVT requires an additional calculation of the step because it depends on N.

3.5.4.2 *Intersection*

Intersection is done by comparing bitmasks

The first function after entering the main loop is IntersectionTest(). It is necessary to compare the current t of the ray with the voxels of a brick, to decide which operations need to be executed in the current iteration. To allow the comparison between ray and brick, it is required to unify their representations. Since the compactness of the brick in a bitmask representation g_{bit} should be preserved, the ray representation is transformed into a corresponding bitmask r_{bit}

Variable	once	stack	calc.	Meaning
$\vec{o}$	x			origin of the ray
t		x		current t on the ray
t_{min}	x			entry point of ray in bounding box
t_{max}	x			exit point of ray in bounding box
$\vec{d}$	x			direction of the ray (always normalized to use t as ray length)
l			x	current hierarchy level
g_{bit}		x		bitmask for representing the brick geometry
id		x		pointer index to first child node
s_{pop}		x		stack for popping to correct parent node
$\vec{c}_{\{min\}}$		x		minimum corner of bbox with respect to $\vec{o}$
$\vec{c}_{\{max\}}$		x		maximum corner of bbox (implicitly given by $\vec{c}_{\{min\}}$ and $\Delta p_{\{x,y,z\}}$)
$\Delta p_{\{x,y,z\}}$	(x)		x	steps between succeeding planes in x, y and z (isotropic voxels need only one value, corresponds to voxel edges)
$v_{\{x,y,z\}}$		x		indices of the current voxel
$\Delta t_{\{x,y,z\}}$	(x)		x	t-steps between succeeding planes
$m_{\{x,y,z\}}$		x		next t for intersecting x-, y- or z-plane
t_{before}			x	t of entry point of current voxel
t_{next}			x	t of exit point of current voxel
$s_{\{x,y,z\}}$	x			steps in index space: either -1 or +1
r_{bit}			x	bitmask for representing current t on ray

Table 3.8: The initialized variables and their descriptions are provided. It is marked if they need to be processed only once for initialization, if they need to be maintained for every hierarchy level in a stack or if they are calculated from stack variables. The marks in brackets show how the variables could be handled alternatively.

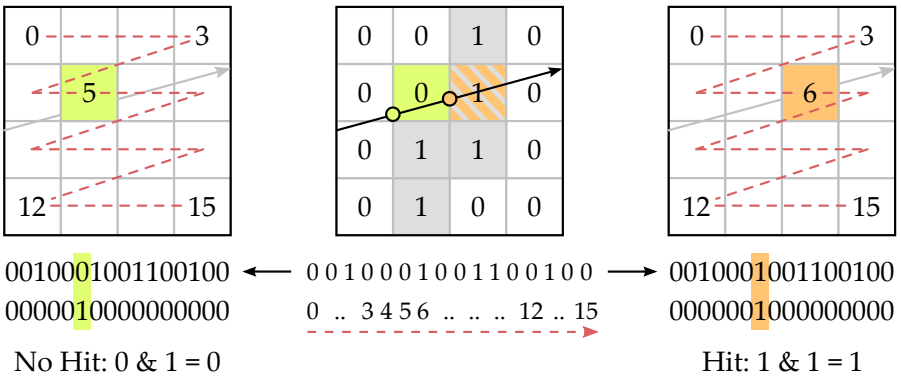


Figure 3.23: The calculation of intersections with bitmasks is shown. The brick in the middle is traversed by the ray and represented with the bitmask g_{bit} at the bottom. Empty voxels are represented with 0 and non-empty voxels are represented with 1. The according bits are sorted by the Morton order (dashed red lines). Left and right, the determination of r_{bit} and the intersection test is shown. The sample on the ray corresponds to a bit in the bitmask. A bitwise AND between g_{bit} and r_{bit} shows, that no hit has been found left, while a hit was found on the right.

by using the indices of the voxel $v_{\{x,y,z\}}$ that is entered by the current t of the ray. $v_{\{x,y,z\}}$ is transformed to a Morton index, which is used for a bitshift in r_{bit}. Afterwards, a bitwise AND operation between g_{bit} and r_{bit} allows to determine an intersection of ray and non-empty voxels in the brick. Fig. 3.23 illustrates the determination of r_{bit} and the intersection.

Differences to [LK10]

The discussed intersection is similar to the intersection test of [LK10], because the geometry and the active voxel are encoded with bitmasks as well. One main difference is the hierarchical treatment of contour intersections in [LK10] (see Sec. 3.3.1), which is only applicable if the traversal is terminating at the first surface. While contours can be turned off in [LK10], this functionality is omitted in the rendering of the proposed SVT, because the main purpose of SVT rendering is the traversal of thick surfaces. Another minor difference is the comparison of varying bitmask sizes for varying number of children in the SVT. While [LK10] always compares 8 bits, the SVT intersection needs to compare up to 125 children, which are aligned to a maximum of 128 bits.

3.5.4.3 *Attribute Determination*

Application of the SVT is defined by attribute determination

The attribute determination is the most important step in respect to the application. While the other functionalities, i.e. initialization, intersection and hierarchical traversal, are fixed, the final output of the SVT rendering depends solely on the criteria for using a voxel attribute and the attribute determination. The two main applications

for the SVT are a surface rendering and a THz simulation. While the application for surface rendering is discussed in this section, the changes in the attribute creation for a THz simulation are discussed in chapter 4.

Colors and normals are extracted for rendering

In case of surface rendering, two stopping criteria are used. While an optional stopping criterion based on the ratio of voxel size and pixel size is discussed in Sec. 3.5.5.2, a necessary stopping criterion is based on the deepest hierarchy level. If the ray intersects a non-empty voxel on the finest level, a further descent is not possible. Therefore, the traversal needs to be terminated and the attribute needs to be determined. In this case, the compressed 32-bit value of the voxel is used for calculating the final output. This value consists of a normal and a color (see Sec. 3.3.3). Bitmasks and bitwise operations (see Lst. 1) are used to extract the R, G, B channels of the color and the compressed normal values $[u, v]$. The compressed normal values need to be processed further by normal-decoding (cf. Eq. 3.4 - 3.6 in Sec. 3.3.3). After the single components are obtained, they are used for a direct rendering with or without a simple local illumination of diffuse material.

```
r = ((attribute >> 27) & 31)/31.f
g = ((attribute >> 21) & 63)/63.f
b = ((attribute >> 16) & 31)/31.f
u = (((attribute >> 8) & 255) - 127.5f)/127.5f
v = ((attribute & 255) - 127.5f)/127.5f
```

Listing 1: Bitoperations for extracting the color channels r,g,b and compressed normal coordinates u,v from the 32-bit attribute value are shown. The attribute value is shifted to represent the individual value in the least significant bits. After extracting the value with a bitwise AND of a bitmask with 1 in all bits that belong to the value and zero in the other bits, a cast to float (omitted in the listing) and a division by the maximum value is done. r,g,b are in the range $[0, 1]$, while u,v are in the range $[-1, 1]$.

Attribute handling of [LK10]

[LK10] proposes a novel compression of normals and a color compression in the spirit of DXT1. Hence, a decompression is needed. For color compression, blocks with 64 bits are used to store 16 voxel colors of two consecutive bricks with 8 possible voxels each. 32 bits store two 16-bit reference colors $\mathtt{col}_1, \mathtt{col}_2$ which are encoded with RGB565 as well. The other 32 bits store two bits for each voxel to represent one of the four fixed interpolation variants between these two main colors ($100\%\,\mathtt{col}_1$, $100\%\,\mathtt{col}_2$, $33\%\,\mathtt{col}_1 + 66\%\,\mathtt{col}_2$, $66\%\,\mathtt{col}_1 + 33\%\,\mathtt{col}_2$). For the normal compression, a similar approach is used. Here, blocks with 128 bits in total are used to store 16 voxel normals. All 16 normals are composed of three vectors which are used to represent a parametric plane equation without the restriction of linearly independent vectors. The directional vectors of the equation are scaled inde-

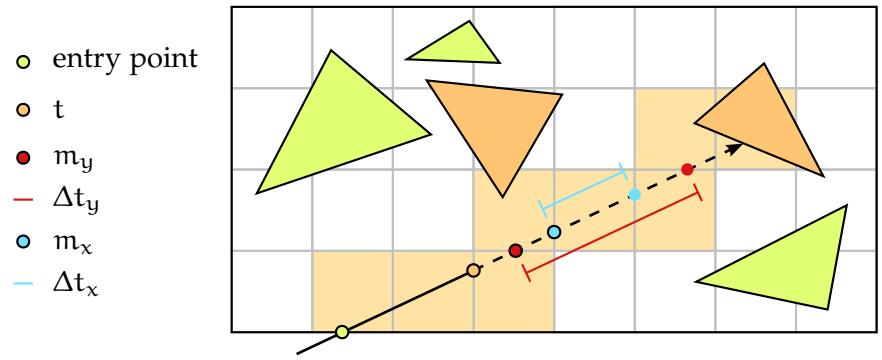


Figure 3.24: The voxel ray traversal of [AW87], which is used by the proposed SVT as well. The ray enters the voxel grid at the green dot. The orange dot shows an exemplary t on the ray. The next t is set to m_y (red dot), because m_y is less than m_x. m_y is set to $m_y + \Delta t_y$ for the next iteration. The orange triangles are tested for intersection. Multiple intersection tests of the same triangle are avoided and correct order of intersections is ensured.

pendently per voxel. 32 bits are used for the support vector and 16 bits are used for one directional vector, which leads to 64 bits. Each voxel stores two scaling values which are multiplied by the two directional vectors. One scaling value has a size of 2 bits and represents 4 fixed values $(-1, -\frac{1}{3}, \frac{1}{3}, 1)$. In comparison to the implemented decompression for SVTs, the decompression of [LK10] is more complex. Therefore, it is assumed that the attribute determination of [LK10] is less performant, but it is negligible for surface rendering, because the traversal stops after the first intersection and the decompression is done only once. For a frequent decompression of traversed voxel attributes in a rendering of thick surfaces, the performance would drop faster for [LK10], but a traversal of several colors values would not be meaningful.

3.5.4.4 *Hierarchical Traversal*

Traversal is done by Push(), Step() and Pop()

To traverse SVOs or SVTs with a stack, it is necessary to provide the three basic functionalities for descending to finer hierarchy levels (Push), ascending to coarser hierarchy levels (Pop) and stepping through voxels on the same hierarchy level (Step).

Traversal of [AW87] is applied in Step()

Step() is performed by a loop for the ray traversal which is proposed by [AW87] (see Fig. 3.24). The main loop is executed until the ray exits the grid or finds an intersection with referenced geometry. In each iteration the minimum of $m_{\{x,y,z\}}$ is used to determine if the ray succeeds to the next voxel by intersecting the subdivision plane in x-, y- or z- direction. After this determination of $\min(m_{\{x,y,z\}})$, the new voxel index $v_{\{x,y,z\}}$ is set and the corresponding $m_{\{x,y,z\}}$ is updated. Alg. 3.6 shows this behavior inside the loop. Afterwards, the referenced geometry of the new voxel is tested for intersection. Here,

Algorithm 3.6 : Loop of [AW87] for voxel grid traversal

```
// Input
t                 // current t on the ray
v_{x,y,z}         // indices of the voxel
s_{x,y,z}         // steps in index space: either -1 or +1
m_{x,y,z}         // next t for intersecting x-,y- or z-plane
Δt_{x,y,z}        // t-steps between succeeding planes in x, y and z

// Traversing the subdivision planes of the voxel grid
while ray is inside grid and no geometry is intersected do
    if m_x < m_y then
        if m_x < m_z then
            v_x ← v_x + s_x
            if v_x outside of voxel grid then
                break
            m_x ← m_x + Δt_x
        else
            v_z ← v_z + s_z
            if v_z outside of voxel grid then
                break
            m_z ← m_z + Δt_z
    else
        if m_y < m_z then
            v_y ← v_y + s_y
            if v_y outside of voxel grid then
                break
            m_y ← m_y + Δt_y
        else
            v_z ← v_z + s_z
            if v_z outside of voxel grid then
                break
            m_z ← m_z + Δt_z
    checkIntersectionWithObjects(v_x, v_y, v_z)
```

it is guaranteed that each triangle is only once tested for intersection by maintaining triangle and ray identifiers. Furthermore, it is ensured that the earliest intersection is found by tracking the smallest t of all intersections. The difference between the rendering of the proposed SVT and the approach of [AW87] is the type of intersection. Instead of intersection tests with triangles inside the voxels of [AW87], only voxels are tested for the SVT.

Push() stores the state of the current brick

If the ray intersects a non-empty voxel and no stopping criterion is fulfilled, the traversal needs to be continued at a finer hierarchy level. Therefore, Push() is executed once per child brick. First, it is ensured that Step() can be executed on the current level if the ray

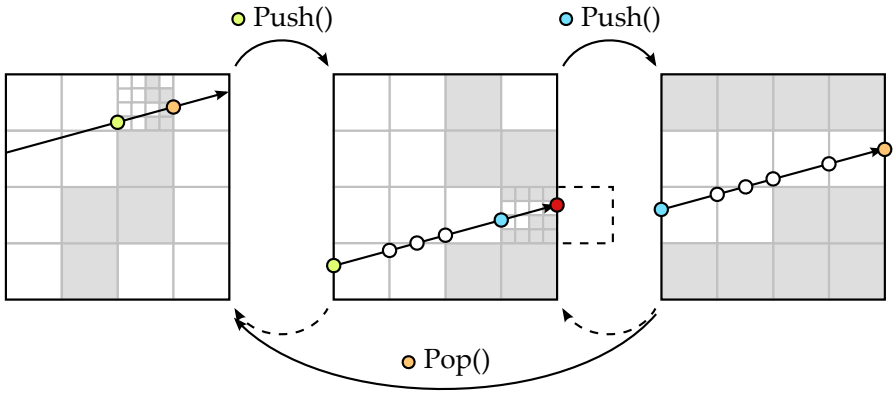


Figure 3.25: The stack-based traversal of a 4^2-tree with 3 hierarchy levels is shown. Push() and Pop() are used to switch between the hierarchy levels. After the first intersection in the coarsest level (green dot, left brick), Push() is storing the state for continuing the traversal on that level (orange dot, left brick). The traversal in the finer level is done by Step() (white dots) until the next intersection is encountered (blue dot, middle brick). Here, the Push() tries to save the state by determining the next voxel (dashed cube), but the level is traversed completely (red dot). On the finest level, no intersection is found and the Pop() needs to be executed for ascending the hierarchy. In the descent, the middle brick was traversed completely, so that Pop() switches directly to the coarsest level again (orange dot).

does not terminate in finer hierarchy levels. By determining the next voxel of the current brick it is decided if the ray will need to traverse this brick further. If the next voxel in the traversal is outside of the current brick, s_{pop} of the current level is marked as "traversed" and no stack variables need to be stored, because the ray will not traverse the brick again. Only if the next voxel is inside the brick, s_{pop} of the current level is marked as "not traversed" and all stack variables are stored (see "stack" column of Tab. 3.8). An example for this behavior of Push() and Pop() is given in Fig. 3.25.

Push() initializes the ray traversal for the next child brick

After storing the state of the parent, the traversal is initialized for the child node. The initialization for the root node (cf. Sec. 3.5.4.1) is applied here in the same manner. Only the first intersection test with `bbox` is omitted, because the ray is inside `bbox`. After this initialization, the next iteration of the main loop is started without executing further operations on the current level.

Pop() determines a coarser hierarchy level for continuing traversal

If the ray traversed a brick without being terminated, it is necessary to use Pop() for determining the hierarchy level where the traversal needs to be continued. While Push() is executed, the traversed bricks of the hierarchical descent are marked as "traversed" or "not traversed" in s_{pop}. Accordingly, Pop() determines the level for continuation by iterating over s_{pop} until the first incompletely traversed brick is found. The stack variables of that level are loaded and the

traversal continues. If Pop() reaches s_{pop} for the root brick and its status is "traversed", the traversal is terminated.

Comparison with stack traversal of [LK10]

The concept of [LK10] for traversing SVOs is used to render the proposed SVT, but the implementation between the proposed rendering and the rendering of [LK10] differs. Since SVOs have only one subdivision per dimension in the brick, the ray can only be in one of two states. Therefore, [LK10] can use efficient binary operations for tree traversal. Instead of maintaining the next t-values for intersecting x-, y- or z-plane ($m_{\{x,y,z\}}$), [LK10] calculates t-values for planes in the center of the brick, compares them against the current t on the ray and flips according bits. With this simplification, it is possible to decide more efficiently whether Push(), Step() or Pop() is needed in the next iteration, because no additional calculation for locating the succeeding voxel is necessary. While the operations of [LK10] can be executed more efficiently, the proposed SVT traversal decreases the total number of operations due to a decreased hierarchy depth.

3.5.5 *Optimizations*

Following subsections address performance optimizations

Performance improvements of the ray traversal are discussed and compared with optimizations of [LK10], which serve as foundation. Mainly, the length of the ray can be optimized to reduce the number of traversed voxels of the SVT. For skipping empty space before the first possible intersection, the rays are shortened by beam optimization (Sec. 3.5.5.1). To terminate the ray traversal by an early exit (Sec. 3.5.5.2), distance-dependent stopping criteria are used. Another optimization is the reduced memory latency for accessing stack variables with the use of local variables which are used as a bitstack (Sec. 3.5.5.3).

3.5.5.1 *Beam optimization*

Beam optimization improves performance by skipping empty space

To ensure that all intersections of ray and voxel geometry are found, the ray traversal starts with the first intersection point on the bounding box which is given by t_{min} or it starts with $t = 0$ if origin $\vec{o}$ of the ray is inside the bounding box. The idea of beam optimization is the estimation of a better start position of the ray to reduce the iterations of the traversal loop, which results in a performance gain. The traversal of empty space between the first non-empty voxel and the ray start position is optimized by shortening the ray.

Beam rays allow shorter ray lengths

The first possible intersection with the voxel geometry needs to be estimated to calculate the improved start position of the ray. If every ray determines this intersection individually, it would represent the normal ray traversal. Therefore, neighboring rays are grouped to beams. Each beam is represented by 4 surrounding rays which traverse the scene in a preceding render pass. They stop after finding an intersection and provide the smallest t of the 4 t-values as starting

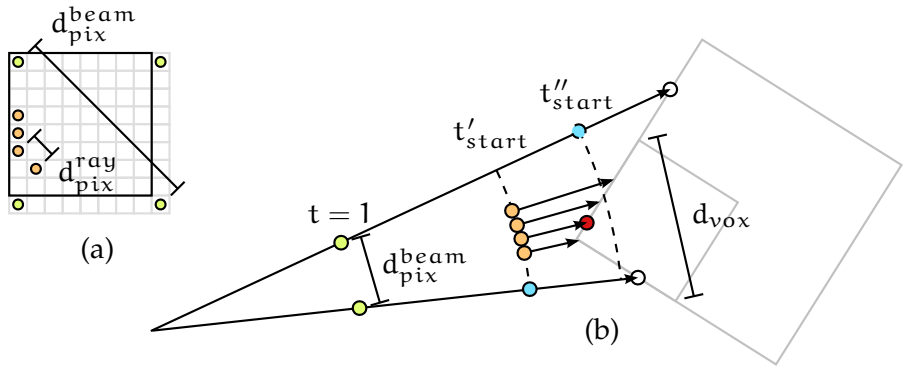
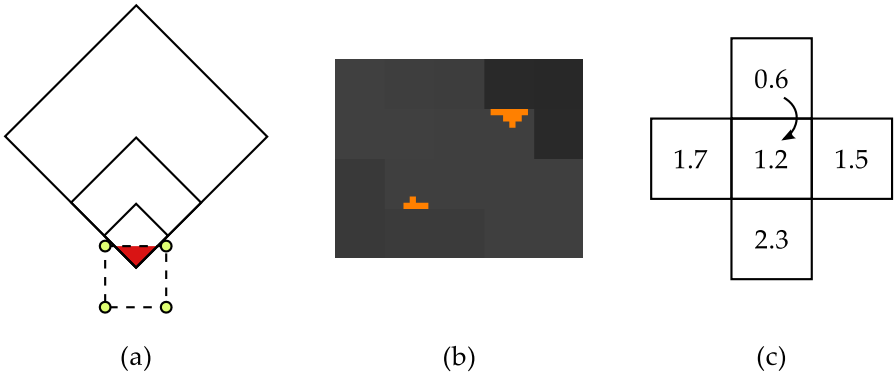


Figure 3.26: The idea of beam optimization from [LK10] and the used parameters of the SVT rendering are depicted. 8x8 pixels (black square) of the final resolution are represented by 4 beam rays (green dots) and the diagonals d_{pix}^{beam} and d_{pix}^{ray} between neighboring rays are shown (a). Please note, that they are calculated in world space (cf. green dots in b). The scene is sampled by beam rays to gain performance in the rendering at higher resolution by improving starting positions of the finer rays (orange dots). The traversal needs to be terminated as soon as some features would be missed when rendering at higher resolution (red dot). d_{pix}^{beam} is projected and compared with d_{vox} to terminate the traversal. Both beam rays find an intersection with the gray voxel at a coarser level and terminate the traversal, because the child voxel with d_{vox} is missed. The minimum of neighboring starting points t'_{start} and t''_{start} (blue dots) is used (b).

point to all rays inside the beam. Due to this coarser ray sampling of the scene, it would be possible to miss small details and the estimation of the first intersection inside the beam would not be usable. Therefore, the stopping criterion needs to take into account a comparison of voxel and pixel.

Stopping criterion for ray termination

A comparable representation of voxel and pixel needs to be found for the termination of the ray. Therefore, the size of the projected voxel on the screen needs to be compared with the pixel size. In case of a common perspective projection, all rays have the same origin and the distance between the rays in relation to the length of the ray t allows to calculate the length of a pixel diagonal d_{pix}^{beam} at the point of voxel intersection. Since the precise calculation of the voxel coverage on the screen would not be performant, because the orientation of the voxel needs to be considered, the voxel diagonal d_{vox} is used to represent the largest possible extent and approximates the voxel size efficiently. The comparison of d_{pix}^{beam} and d_{vox} allows to determine the ratio of pixel size to voxel size. While the traversal is performed, d_{pix}^{beam} increases due to an increasing t and d_{vox} varies due to different hierarchy levels. Therefore, it is necessary to do the comparison in each iteration.



(a) (b) (c)

Figure 3.27: The special case for a failing beam optimization is shown. If the beam rays do not intersect voxels in the coarse hierarchy, the stopping criterion cannot be tested (a). A wrong t_{start} is calculated and possible features are missed. The gray values show the calculated t_{start}-value per beam (8x8 pixels). In the case of the orange pixels, an intersection was found in front of t_{start} and features are missed (b). In the proposed dilation, every t_{start}-value is compared with the direct horizontal and vertical neighbors and replaced, if a smaller t_{start}-value is found (c).

Comparison between pixel and voxel size

To terminate the ray before small voxels could be missed, the largest possible pixel size needs to be compared with the smallest possible voxel size of the next step, i.e. the edge length of the child voxels Δp^{child}. The largest possible pixel size is determined by the sum of ray length at the voxel exit point t_{next} and $\frac{d_{vox}}{2}$ which compensates a voxel orientation in the worst case (cf. dashed lines in Fig. 3.28). The comparison is shown in Eq. 3.14. If Eq. 3.14 is fulfilled, the ray is terminated and the optimized start value t_{start} for all rays of the beam is calculated by Eq. 3.15. Since the voxel is arbitrarily oriented to the ray direction and a corner of the detected voxel can be nearer than the intersection point, $\frac{d_{vox}}{2}$ is subtracted from the voxel entry point t_{before}.

$$d_{pix}^{beam} \cdot (t_{next} + \frac{d_{vox}}{2}) > \Delta p^{child} \tag{3.14}$$

$$t_{start} \leftarrow t_{before} - \frac{d_{vox}}{2} \tag{3.15}$$

Special case of missed voxel corners needs to be considered

This processing is possible due to the hierarchical tree structure of SVOs and SVTs. Since every non-empty child node has a non-empty parent node, no small voxel is missed, because the ray first passes the coarser parent node and the stopping criterion ensures an early termination on the coarser level. One special case is encountered, if a corner of the parent voxel lies between two coarse rays and a non-empty voxel in a finer hierarchy level lies in this corner as well (see Fig. 3.27 (a)). Here, the stopping condition can not be tested, because

the coarser level is missed, which leads to an error in the estimation. Fig. 3.27 (b) shows a screenshot of the resulting error. While gray pixels show that the intersection is behind the estimated t_{start}, the colored pixels show intersections in front of t_{start}.

Dilation is proposed to correct the resulting error

To avoid this behavior, the resulting array of t-values is modified before being passed to the traversal. In the special case of missed corners, neighboring rays hit the missed parent voxel and estimate a correct t_{start} due to the special voxel orientation and the stopping criterion. Therefore, an additional dilation step is proposed. Each t-value is compared with its direct neighbors in the left, right, top and bottom direction of the image plane. The minimum of these 5 t_{start}-values is written to the final array of optimized t-values for the ray traversal (see Fig. 3.27 (c)). While this method provides a correct estimation, the more conservative t_{start}-values lead to a less efficient beam optimization due to more potential steps through the hierarchy.

Difference to [LK10]

In comparison to the proposed beam optimization, [LK10] does not use the mentioned dilation step and encounters the shown error for missed voxel corners. Another difference is the efficiency of the beam optimization. The stopping criterion is less efficient for SVTs, because the hierarchy depth is already reduced and a comparison of voxel diagonals between two consecutive levels has a ratio of $\frac{1}{N}$ instead of $\frac{1}{2}$ which leads to an earlier termination of the ray and a more conservative t_{start}. Although the beam optimization improves the performance, the proposed ray traversal does not benefit from the beam optimization as [LK10] does due to less precise approximations between levels and the additional dilation.

3.5.5.2 *Early exit*

Advantages of early exit

The standard traversal of SVTs is terminated if the ray intersects a non-empty voxel on the finest hierarchy level or if it reaches t_{max} and leaves the bounding box of the voxel grid. To save computation time and to improve the visual quality, the traversal can be terminated earlier. While computation time is reduced by less iterations of the main traversal loop, the visual quality is improved if an undersampling of fine details is avoided. This undersampling occurs if the distance between voxels is smaller than the distance between rays so that the sampling theorem is harmed. In this case, the high spatial frequency represented in the voxel grid cannot be restored by the sparser sampling of the rays as determined by the image resolution. Therefore, random colors and noise patterns appear in the rendering.

Smallest possible pixel size is compared to largest possible voxel size

In principle, pixels and voxels are compared with the same diagonals as discussed in Sec. 3.5.5.1 and shown in Fig. 3.26, because the ray needs to be terminated earlier as well. The main difference between beam optimization and early exit is the purpose. While the coarse render pass of the beam optimization needs to be terminated before a smaller voxel will be missed, the early exit can only be done,

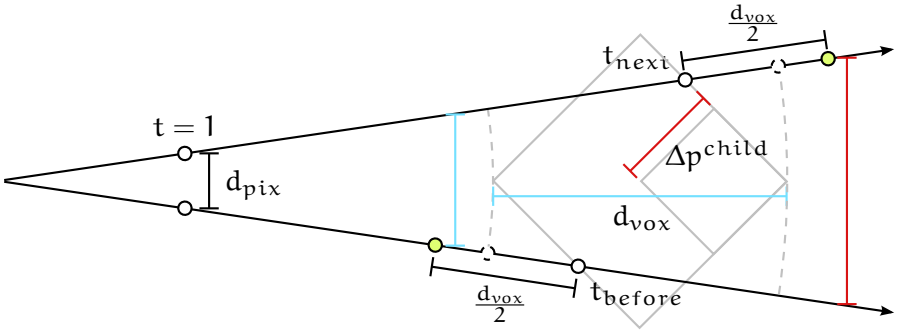


Figure 3.28: The stopping criteria for beam optimization and early exit are depicted. Pixel distance d_{pix} is scaled by $t_{before} - \frac{d_{vox}}{2}$ to compare it with the voxel diagonal d_{vox} for the early exit (blue lines). For the beam optimization, d_{pix} is scaled by $t_{next} + \frac{d_{vox}}{2}$ to compare it with edge length of the child voxel Δp^{child} (red lines). In both cases, $\frac{d_{vox}}{2}$ is used to compensate the varying voxel orientation, which influences the calculation of the extreme pixel distances (gray dashed lines).

if the intersected voxel is smaller than a pixel. To incorporate these requirements, Eq. 3.16 is used to terminate the traversal. Please note, that d_{pix}^{ray} has the same meaning as d_{pix}^{beam}, but is smaller due to a higher resolution, i.e. a denser ray sampling of the scene. The smallest possible pixel size is calculated with a shift of t_{before} by $\frac{d_{vox}}{2}$ in the direction of the ray origin. Furthermore, the diagonal of the intersected voxel d_{vox} on the current hierarchy level is used to obtain the largest possible voxel size. Fig. 3.28 depicts the differences between the stopping criteria of beam optimization and early exit.

$$d_{pix}^{ray} \cdot (t_{before} - \frac{d_{vox}}{2}) > d_{vox} \tag{3.16}$$

Early exit requires attributes in coarser hierarchy levels

The early exit of the ray requires attributes on all hierarchy levels of the SVT, because a termination on finest hierarchy level can not be ensured with a stopping criterion that is based on distances. Therefore, the attributes of the fine levels need to be distributed to the coarser hierarchy levels. In the SVT creation (see Sec. 3.4.5.2), attributes of the child nodes can be used to calculate sophisticated parent attributes order-dependently, but for the application of surface rendering, an order-independent averaging of normals and colors is used.

N influences visual appearance

While a parent node in an SVO is averaged by a maximum number of 8 child nodes, the contributing child values for a parent in an SVT depend on the choice of N. An increasing N leads to a larger number of color and normal values which contribute to the attribute of the direct parent. Hence, the individual child attributes have a smaller influence on the parent value and the resulting attribute values are more reliable for larger N from a statistical point of view. In respect of visual quality, a smaller N is more beneficial, because additional hierarchy levels with smaller changes in the attributes lead to smoother

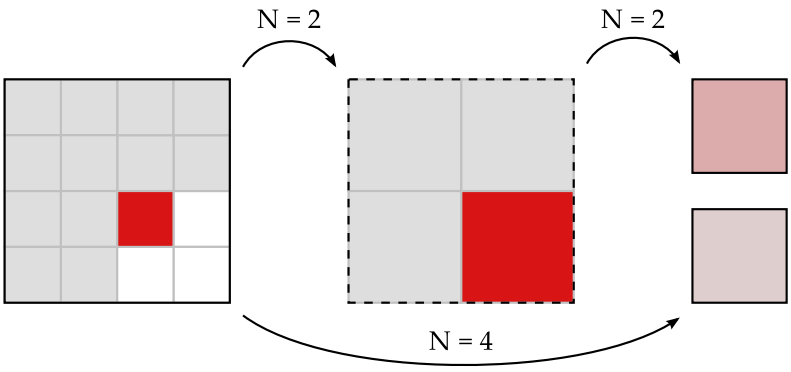


Figure 3.29: The influence of N to the attributes of the coarser hierarchy levels is depicted. While all voxels are averaged directly to the parent attributes for N = 4, an additional hierarchy level for N = 2 is leading to another parent attribute in the coarsest level. If the ray is terminated earlier, the transition between parent and child is smoother with a low N due to more hierarchy levels.

transitions between parent and child. It follows that a rendering of a scene with early exit and varying N is not consistent, because of statistical properties and ray terminations on different hierarchy levels. An example is given in Fig. 3.29.

Visual artifacts

Another influence on the visual appearance is the adaptive ray traversal which skips empty voxels. The stopping criterion can only be evaluated at intersections with full voxels, because empty voxels do not store attributes for calculating an output color. Since the samples on the ray are defined by the scene description, it can not be ensured at which ray positions the stopping criterion is evaluated. It follows, that the correct position of the stopping criterion will rarely be hit and each ray stops individually depending on the intersected voxels in different hierarchy levels. An increasing N amplifies this effect, because of the larger scene complexity inside one brick due to more child voxels and the ratio of voxel diagonals between two hierarchy levels. While the scene complexity inside a brick influences whether the ray terminates generally, the ratio of $\frac{1}{N}$ between voxel diagonals leads to larger distances on the ray when the next hierarchy level might fulfill the stopping criterion. Therefore, a kind of level-of-detail popping is perceivable. Fig. 3.30 shows an example.

Difference to [LK10]

By comparing the voxel size and the pixel size, [LK10] is performing an early exit as well. The ray is terminated if a leaf voxel is found or if the intersected voxel becomes smaller than a pixel. Due to the used contour slabs (see Sec. 3.3.1), the SVO has a locally varying tree depth as well. To compensate the resulting artifacts between different hierarchy levels, the use of a blurring filter is proposed as a post-processing step on the image. A varying radius for sampling and weighting neighboring colors on the image is determined by the size of the voxel. If the voxel is smaller than a pixel, no neighboring

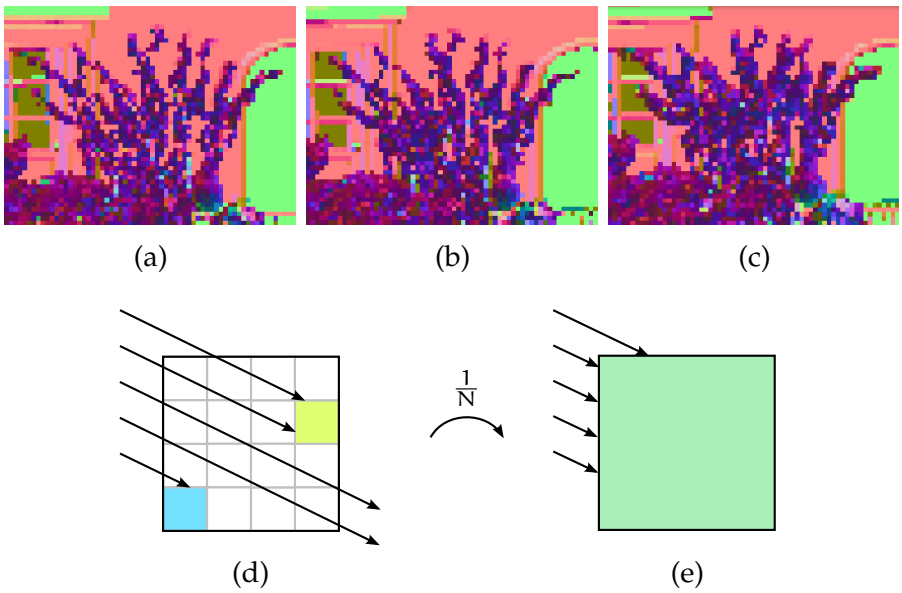


Figure 3.30: The early exit leads to a kind of level-of-detail popping, which is influenced by the skipping of empty voxels and N. A magnified rendering of fine features at increasing distances between scene and eye position is shown from left to right (a - c). While finer details can be seen, if the child voxels are rendered (a), the parent voxels become smaller than a pixel and provide the averaged color of the whole brick (c). Rays, which skip empty child voxels (d), may be terminated if the parent voxel fulfills the stopping criterion by moving it away from the eye position (e). The distances on the ray for a fulfillment of the stopping criterion are influenced by the ratio $\frac{1}{N}$ between voxel diagonals of succeeding hierarchy levels.

pixels are sampled. Although a level-of-detail popping should be visible here as well, it is less obvious due to a small N (cf. Fig. 3.30 (d, e)) and the blurring in image space at the pixel of the coarser hierarchy level, which smooths its color value by sampling adjacent pixels of the finer hierarchy level.

3.5.5.3 *Bitstack*

Reduced memory requirement leads to performance gain

The stack variables of the traversal require a frequent access for reading and writing memory. It leads to memory access latencies, if those accesses can not be hidden by more independent arithmetic operations, which can be executed during the delay time, or an increased workload due to a larger number of concurrent threads. The number of concurrent threads can not be increased further, because the available memory is limited by the large memory consumption of the stack variables per thread. Only a large reduction of needed memory per thread would allow to increase the number of concurrent threads. Since the number of necessary arithmetic operations is optimized al-

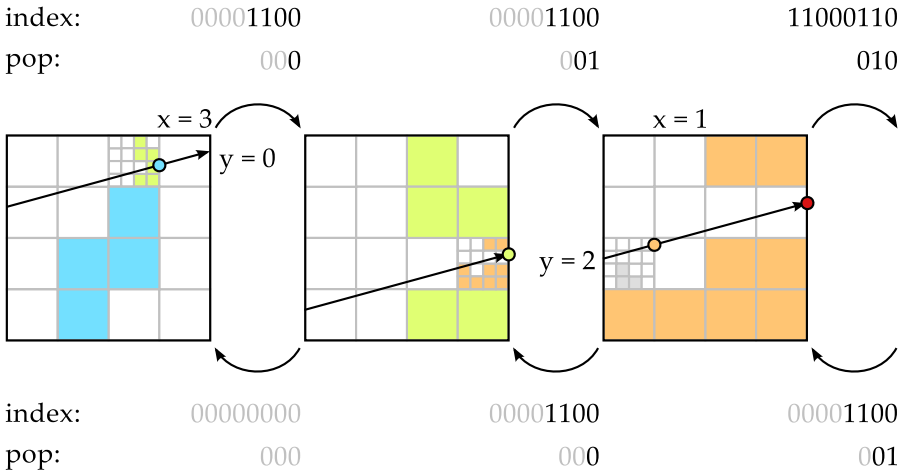


Figure 3.31: An example for the use of bitstacks is provided. The bitstack stores necessary information for continuing the traversal after returning from finer hierarchy levels. It consists of an index stack and a pop stack. The stack state for descending is shown at the top, while it is shown for ascending at the bottom. If the brick is not traversed completely (blue and orange brick), the binary representation of the cube indices $v_{\{x,y\}}$ (2D in this example) is pushed to the index stack and a zero bit is pushed to the popstack. If the brick is traversed completely (green brick), only the popstack is used to store a one bit. In the ascent, the zero bits in the popstack determine the next hierarchy level, where the traversal needs to be continued and $v_{\{x,y\}}$ are popped from index stack.

ready, the arithmetic complexity is low, so that no operations can be executed during the waiting time and the latency can not be hidden as well.

Single bits of one variable represent a stack

Therefore, a bitstack in the spirit of [ÁSK14] is used to reduce the memory requirements and memory accesses of each thread. Furthermore, the values are obtained faster. The bit representation of a variable is used to represent the whole stack. The stack is maintained by bitshifts which push and pop the values into the variable. In the proposed SVT rendering, the cube indices $v_{\{x,y,z\}}$ and the popstack s_{pop} are represented with bitstacks. Fig. 3.31 depicts the principle of both bitstacks.

Popstack avoids unnecessary memory accesses

As stated in Sec. 3.5.4.4, s_{pop} stores whether a hierarchy level is "traversed" or "not traversed" before descending to a finer hierarchy level. This operation is done by performing a bitshift of one bit in the direction to the most significant bit and setting the least significant bit of s_{pop} to 0 or 1. 0 represents the state "not traversed", while 1 means "traversed". Only in case of "not traversed", the other stack variables are stored. For the ascending to a coarser hierarchy level, an iteration over the least significant bits of s_{pop} is performed. The least significant bit is checked and removed by a bitshift to the least significant

bit. This checking is stopped as soon as a hierarchy level is found, which is "not traversed". All other stack variables are loaded and the traversal continues. This procedure allows to gain performance and reduce memory accesses by omitting the processing of stack variables which will not be needed in further iterations.

Voxel indices are stored in one variable

The tracking of the current position of the ray inside the SVT is done by cube indices $v_{\{x,y,z\}}$. For each hierarchy level this index allows to address the voxel of the brick which is intersected by the current t on the ray. To represent those indices with a bitstack in one variable, the number of required bits for a single index are determined by $i_{shift} = \lceil \log_2(N) \rceil$. If the hierarchy level needs to be traversed again, v_x, v_y and v_z are stored as usual binary representation into one variable. Each element is written by a bitshift of i_{shift} and a summation to store all three indices in consecutive bits. In the ascending they are extracted by a bitmask i_{mask}, which is calculated by $2^{i_{shift}} - 1$, and bitshifts in the reverted order. Lst. 2 shows an example.

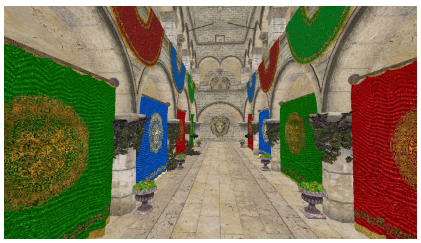
```
N = 5
v_x = 4 // 100
v_y = 3 // 011
v_z = 1 // 001
indexShift = ceil(log(N)/log(2)) // 3
stackVar = 0
indexMask = pow(2,indexShift) - 1 // 7 (0000000111)


// push()
stackVar = (stackVar << indexShift) + v_x // 0000000100
stackVar = (stackVar << indexShift) + v_y // 0000100011
stackVar = (stackVar << indexShift) + v_z // 0100011001

// pop()
v_z = stackVar & indexMask // 0000000001
stackVar = stackVar >> indexShift // 0000100011
v_y = stackVar & indexMask // 0000000011
stackVar = stackVar >> indexShift // 0000000100
v_x = stackVar & indexMask // 0000000100
stackVar = stackVar >> indexShift // 0000000000
```

Listing 2: Bitoperations for pushing and popping the cube indices to a bitstack are shown in an example. The resulting bitmasks after the operation are shown in the comment

Difference to [LK10]

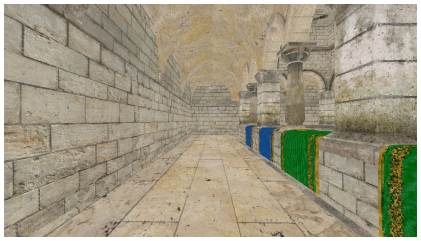
[LK10] uses a bit-representation for storing $v_{\{x,y,z\}}$ in a stack as well, but every dimension is stored in a separate variable. Since, the SVO has only two children per dimension, i_{shift} is not needed and one bit is sufficient to represent the hierarchy level of one dimension. Therefore, the stack is more efficient, because 3 bits are always suf-

Hairball 1

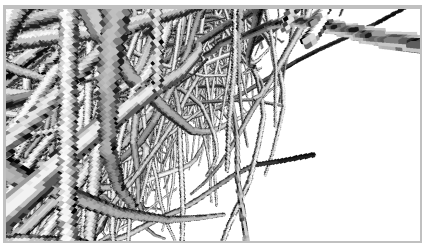


Sponza 1

Hairball 2



Sponza 2



Hairball 3

Sponza 3

Figure 3.32: The shown views are used to evaluate the rendering of SVTs depending on different parameters and applications. To clearly show the Hairball frames, grey contours surround the Hairball images.

ficient for storing $v_{\{x,y,z\}}$ of one hierarchy level. Furthermore, the constant size of one bit allows to interpret the binary representation as integer, e.g. $1_x0_y1_z$ represents 5. This interpretation serves as a referencing to child attributes by relative offsets. In addition, this bit representation allows to calculate the right hierarchy level for continuing the traversal when Pop() is executed. Hence, a separate s_{pop} is not used.

3.5.6 *Results*

Rendering of SVTs is evaluated in several scenarios

To evaluate the rendering, the SVT is applied to common surface rendering and volume rendering. Afterwards, the potential of thick surface rendering for a THz simulation is analyzed. A comparison with specialized techniques for surface and volume rendering is less meaningful, because those techniques are superior due to the exclu-

sive focus on only one of the mentioned applications. Nonetheless, a comparison with the surface rendering of [LK10] provides an indication for the applicability of SVTs in a surface rendering. The following sections focus on a detailed evaluation of parameters and the influence of N.

Description of evaluation environment

All tests have been performed on an Intel Core i7-930 (2.80GHz) with 24 GB of RAM and an NVIDIA GeForce GTX 680 with 2 GB of memory. The scenes Hairball at a resolution of 2048^3 and Sponza at a resolution of 4096^3 are used. 3 views for each scene with different complexities and properties have been taken. Fig. 3.32 shows these views. While Sponza is showing voxel colors from diffuses textures of the original mesh, Hairball is rendered with a local illumination of diffuse light.

3.5.6.1 *Surface Rendering*

[LK10] and SVT are compared by omitting contours

COMPARISON WITH [LK10] The use of SVTs in a surface rendering is obvious due to similarity to the SVO of [LK10] which provides a very performant surface rendering of voxels. Since [LK10] focuses on surface rendering, performance optimizations are used which are not suitable for thick surface rendering or volume rendering. Here, the main optimization is the use of contours which allows to improve the performance by an earlier ray termination if the original surface mesh is approximated good enough (see Sec. 3.3.1). To provide a fairer comparison with SVT rendering, contours are omitted. Therefore, the performance numbers do not reach the highest possible values but they provide a better relative comparison to SVT rendering.

Influence of ray incoherence

Tab. 3.9 shows the measurement. All measured values are given in Million rays per second. The 6 views in Fig. 3.32 are rendered at three resolutions (768x432, 1024x576, 1280x720) with early exit enabled. Measurements with and without beam optimization are shown. For the SVT only the most performant choice of N is provided. It can be seen that the rendering of [LK10] is more performant than the SVT rendering. It reaches a factor of up to 1.8 in comparison to the performance of the SVT. In two measurements, the SVT reaches the same performance as the SVO. These are the most performant measurements of the chosen views. It follows that the scene complexity has a larger influence on SVT rendering than on SVO rendering. The scene complexity is directly influencing the possibility of ray coherence, i.e. more rays are executing the same operations per clock cycle. Although both approaches benefit from this ray coherence, the performance of SVT rendering drops stronger for an increasing ray incoherence. This effect can be seen by varying resolutions as well. If the scene is shown at a larger resolution, the frame rate gets lower but the number of processed rays increases due to more coherent rays which render the same area on the rendered surfaces.

		[LK10]		SVT		
		w/o beam	beam	w/o beam	beam	N
Hairb. 1	1280x720	38.7	42.2	24.8	25.9	4
	1024x576	41.9	45.5	24.3	25.0	4
	768x432	41.1	44.0	25.4	26.0	3
Hairb. 2	1280x720	42.2	58.0	32.9	37.0	3
	1024x576	39.4	51.4	30.5	32.5	3
	768x432	36.3	44.8	27.3	29.5	3
Hairb. 3	1280x720	42.1	68.0	36.4	46.1	4
	1024x576	39.6	60.1	33.5	39.6	4
	768x432	36.5	51.3	29.6	33.4	4
Spon. 1	1280x720	53.9	71.7	47.0	55.8	4
	1024x576	52.3	67.0	45.6	52.6	4
	768x432	50.9	63.8	41.7	46.6	4
Spon. 2	1280x720	69.7	90.1	70.2	90.6	3
	1024x576	67.5	85.2	66.7	80.6	3
	768x432	64.9	80.2	57.7	68.1	3
Spon. 3	1280x720	101.1	130.1	102.9	130.6	4
	1024x576	97.1	124.5	98.7	121.6	4
	768x432	91.5	114.4	93.3	112.2	4

Table 3.9: The rendering performance of [LK10] and the SVT is compared by 6 different views at 3 different resolutions. Performance with and without beam optimization is shown. Depending on N, the most performant SVT rendering is provided. The corresponding N is given in the right column. The best performance per row is shown in bold. All values are given in Million rays per second (Mray/s).

Generalization of SVTs is slower than specialized SVOs

An additional reason for a lower performance of the SVT is the possibility to choose N and generalize the number of children per node of the voxel tree. For this reason, calculations inside the ray processing need to be more general, while the SVO with N = 2 of [LK10] can use more efficient bitwise operations (see Sec. 3.5.5.3).

Influence of N and optimizations is evaluated on different views

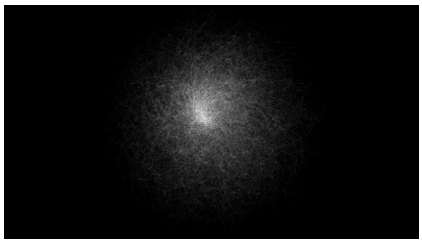
OPTIMIZATIONS To evaluate the influence of N and the optimizations on the rendering (see Sec. 3.5.5.1 and 3.5.5.2), N varies between 2 and 5 for rendering the 6 views. Tab. 3.10 shows the measured results. For each measurement, beam optimization and early exit are enabled. The performance for using only beam optimization or only early exit is given as well. The best performance values using early exit have already been shown in the comparison with [LK10] (see Tab. 3.9).

		Ha. 1	Ha. 2	Ha. 3	Sp. 1	Sp. 2	Sp. 3
N = 2	both	21.3	30.0	39.1	40.5	55.2	97.1
	early	20.6	27.4	29.3	33.3	45.9	75.0
	beam	20.2	30.3	39.8	39.1	55.8	99.7
N = 3	both	25.5	37.0	44.9	54.8	90.6	108.8
	early	24.0	32.9	33.6	47.9	70.2	92.6
	beam	26.1	38.0	46.1	55.3	93.1	114.3
N = 4	both	25.9	36.1	46.1	55.9	75.6	130.6
	early	24.9	32.9	36.4	47.0	64.9	102.9
	beam	26.6	37.2	47.4	55.8	76.5	134.1
N = 5	both	23.6	33.1	40.2	52.8	59.8	99.0
	early	21.7	29.2	32.2	48.0	50.2	87.0
	beam	25.9	36.3	43.5	58.0	65.2	109.0
slowest/fastest		0.76	0.72	0.62	0.57	0.49	0.56

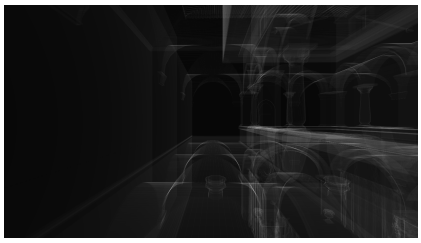
Table 3.10: The performance of early exit and beam optimization is evaluated by rendering the views with different settings. The three settings are: *both* optimizations are switched on, only *early* exit is used, only *beam* optimization is used. The resolution is set to 1280x720. N varies between 2 and 5. The most performant rendering per view is marked in blue. For all views except Hairball 1, the worst performance is obtained by N = 2 without beam optimization. For Hairball 1, the worst performance is obtained by N = 2 without early exit. The last row shows a factor between best and worst performance. A factor of 1 would mean that both values are equal.

Memory access latency is the bottleneck

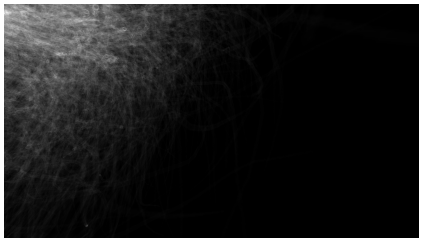
The direct comparison of the optimizations shows, that the single use of early exit leads to the lowest performance independently of N or the rendered view. Intuitively, this is not reasonable because less calculations are required if the rays terminate earlier. Here, the main drawback of generalized SVTs becomes apparent. Due to universal calculations and more effort for processing intermediate results depending on N, the memory footprint of a single thread increases. This memory footprint is linked to the number of registers per thread which is limited to GPU-specific properties. If all frequently accessed variables are stored in registers, the memory access latency is low. All variables which do not fit into registers need to be stored in local memory, where the latency is higher and a register spilling is occurring. In case of early exit, additional variables are necessary to maintain distance criteria for an early termination of the rays. Since the threads have a high memory consumption already, the additional distance values can not be stored in registers and the higher access latency can not be compensated by a reduced computation effort.



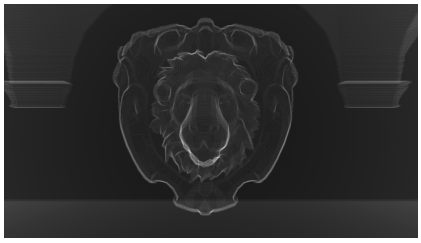
Hairball 1



Sponza 2



Hairball 3



Sponza 3

Figure 3.33: The 4 shown views are used to evaluate the performance of volume rendering with SVTs. The view properties vary in coherence of surface structures and number of iterations. Black pixels represent no intensity, i.e. no voxels were hit. White pixels represent the maximum intensity of each view.

Rendering with beam optimization is most performant

Furthermore, the measurement shows that the single use of beam optimization is the most performant method for rendering all test scenes except Hairball 1 and Sponza 1 with N = 2. The scene complexity influences the efficiency of the beam optimization because flat surfaces can be approximated much better by a coarser distance grid than strongly varying surfaces. For these surfaces, the final rays still need to traverse a strongly differing number of voxels, which results in less efficient ray coherence.

Most often the choice of N = 4 is optimal

In addition, it can be seen that the best performance (blue cells in Tab. 3.10) is obtained by different values of N. Therefore, the scene complexity influences the optimal N and the deviation between worst and best performance. While N = 2 never leads to the optimal performance, the other values of N lead to the best performance at least once. Interestingly, N = 4 results in the best performance for the most complex scene (Hairball 1) and the simplest scene (Sponza 3), so that the choice of N for achieving the best performance is not obvious. Since N = 4 is optimal in half of the test cases and near the optimal performance with other values of N, it seems to be the optimal choice.

3.5.6.2 *Volume Rendering*

SVTs allow a precise and adaptive volume sampling

SVTs can be used for a volume rendering if the voxels are interpreted as volumes. The contribution of a voxel to the ray can be determined

exactly by considering the length of the ray between the entry and exit point of the voxel. It follows that the SVT offers a precise and adaptive volume sampling if the ray traversal is not terminated earlier. Therefore, the potential of SVTs for a common volume rendering is evaluated by interpreting Hairball 1, Hairball 3, Sponza 2 and Sponza 3 of Fig. 3.32 as volumes. To achieve a simple volume representation of these surface models, only the lengths of the ray that traverse full voxels are accumulated and interpreted as intensity. For each view the maximum traversed ray length of all rays serves as a normalization factor to represent the complete intensity range. Fig. 3.34 shows the visualization result.

Using SVTs for volume rendering can be meaningful

The performance of these renderings is given in Tab. 3.11. All scenes were rendered with varying N between 2 and 5. Independent of N, it can be seen that the performance dropped by more than an order of magnitude in comparison to the performance for surface rendering (cf. Tab. 3.9 and 3.10). It is obvious that the rendering of SVTs does not reach the performance of interactive volume rendering techniques. Even if the performance is low, it is possible to obtain unbiased volume representations which are not smoothed by interpolation and show the exact thickness of voxels. This precision is only limited by the volume resolution and the density of the ray sampling. Therefore, volume rendering with SVTs can be meaningful if applied to simulation scenarios which require a high precision.

Main influences are iteration count and ray coherence

Further statistics in Tab. 3.11 show that the high iteration count is the main influence on the performance drop. One iteration consists of different basic functions like Push(), Step() or Pop() (cf. while-loop in Alg. 3.5) and does not show the number of individual basis function calls. Therefore, the iteration count serves as a general indication for the workload. The lowest number of iterations is achieved with N = 3, but N = 4 has the higher performance. One reason is that the iteration counts are not necessarily of equal workload, e.g. an iteration with a single Step() just needs a small fraction of instructions which would be needed by a Push() or an AttributeCalculation(). Another reason is the scene complexity which leads to an even larger ray incoherence because the rays traverse the complete volume and threads are executed with less parallelism the further the ray travels.

N leads to differing expenses for iterations

The Hairball 3 with N = 2 has the lowest performance and Sponza 3 with N = 4 has the highest performance. Under the assumption that the iteration counts have equal workloads, a tripling of the iteration count leads to a drop in performance by an order of magnitude. While the averaged number of iterations is quadrupled, the maximum number of iterations is scaled by an order of magnitude as well. It follows, that a clear indication for the performance behavior is not easy to obtain, because several parameters have strongly differing influences on each other. It seems that N = 4 is the best choice, but the values of N = 5 are very similar. One reason might be that N =

		Hb. 1		Hb. 3		Spon. 2	Spon. 3
N = 2	max	2073	51.0%	3011	29.0%	998	275
	mean	243.4	496.4	441.6	622.0	277.6	141.9
	std	384.6	419.7	601.3	630.2	134.5	29.8
	total	224.3 M		407.0 M		255.9 M	130.8 M
	perf	1.82		1.27		2.25	6.12
N = 3	max	1745	49.4%	2476	27.6%	860	258
	mean	218.9	432.8	393.7	543.8	259.1	145.5
	std	340.4	369.2	520.7	541.1	123.0	31.7
	total	201.8 M		362.8 M		238.8 M	134.1 M
	perf	2.56		1.82		3.01	7.91
N = 4	max	1712	46.5 %	2396	27.1%	844	255
	mean	226.0	422.6	396.2	543.7	259.9	134.4
	std	341.9	368.1	516.3	534.3	120.2	28.0
	total	208.3 M		365.1 M		239.5 M	123.9 M
	perf	3.12		2.26		3.83	12.17
N = 5	max	1691	43.7%	2349	27.49%	861	250
	mean	234.7	417.0	411.7	567.7	264.2	136.5
	std	352.4	380.2	521.6	535.4	124.1	27.9
	total	216.3 M		379.4		243.5 M	125.8 M
	perf	3.04		2.11		3.63	10.39

Table 3.11: To evaluate the performance of volume rendering with SVTs, 4 views with varying complexity are measured. For varying N, statistics of the iterations are shown. The maximum, mean and standard deviation for all pixels of a frame are provided. Since the Hairball views have a lot of pixels with zero iterations (see percentages), statistics for pixels with and without zero iterations are given. Furthermore, the total number of iterations and the performance in Mray/s are provided. The blue cells mark the lowest number of iterations and the highest performance per scene.

5 executes less Push()- and Pop()-operations so that the iterations are cheaper and can be processed faster (cf. Fig. 3.2 in Sec. 3.2).

3.5.6.3 *THz Simulation*

Thick surface rendering is evaluated

Since the main purpose of SVTs is the improvement of THz simulations, a thick surface rendering is required. This rendering allows to incorporate properties of THz radiation regarding penetration depth and attenuation inside real-world materials. The behavior of the ray traversal needs to be evaluated, if it proceeds until a specific ray dis-

tance inside of full voxels is reached. Therefore, the views Hairball 3 and Sponza 2 are rendered by varying ray lengths as stopping criteria. First, the maximum ray length of the traversal through the complete volume is determined per view. Afterwards, this maximum ray length serves as a normalization factor so that each ray can be terminated if it reaches a fraction of this factor, i.e. each ray terminates after traversing a fixed distance through full voxels. The factor

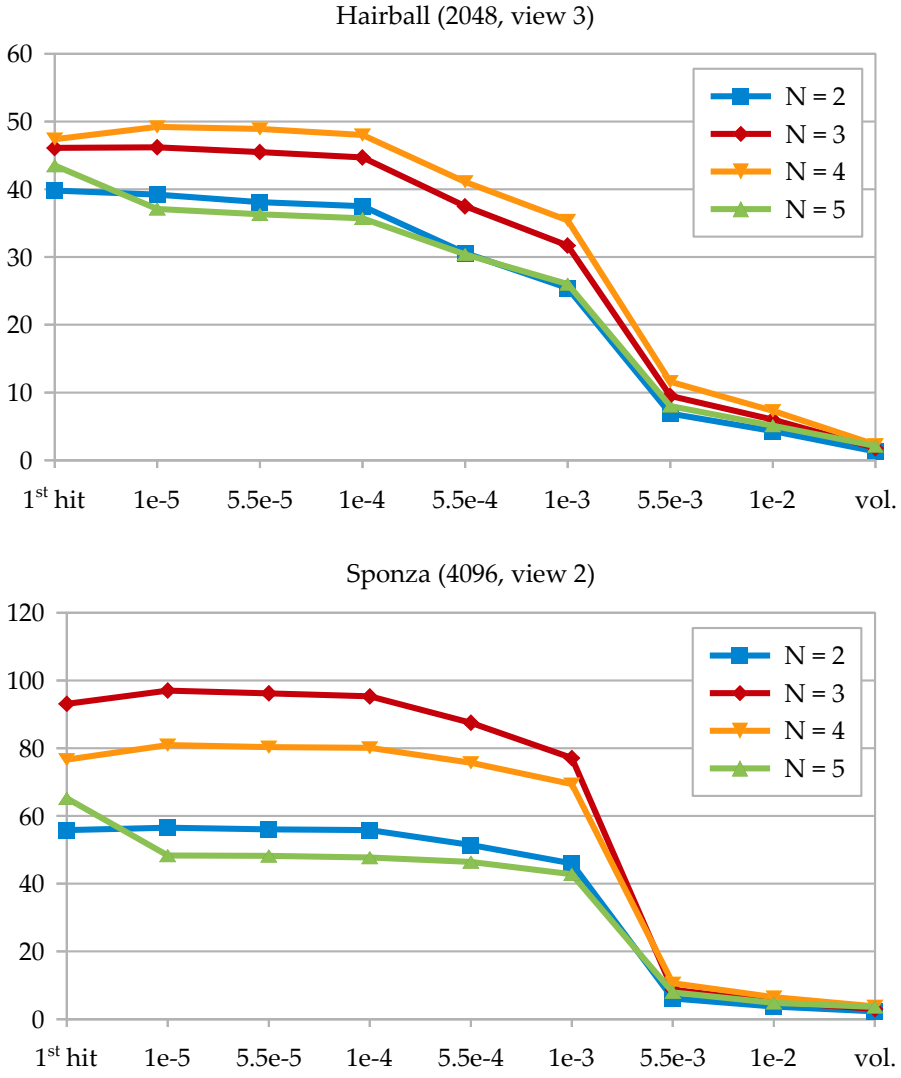


Figure 3.34: To evaluate the feasibility of a thick surface rendering for THz simulations, the performance [in MRays/sec] of the ray termination with different penetration depths is evaluated for Hairball view 3 and Sponza view 2 with varying N. From left to right, the ray needs to traverse more full voxels until it is terminated. For comparison, surface and volume rendering are provided again. While surface rendering terminates the ray after the first hit, volume rendering is traversing the complete scene without terminating earlier.

		Hairball view 3			Sponza view 2		
		1e-3	5.5e-3	1e-2	1e-3	5.5e-3	1e-2
N = 2	max	451	749	933	263	408	525
	mean	56.1	165.9	239.9	29.7	135.2	195.1
	total [in M]	36.7	108.5	157.0	27.4	124.6	179.8
N = 3	max	451	647	784	231	392	529
	mean	53.3	151.6	216.5	20.1	120.3	177.8
	total [in M]	35.6	101.1	144.4	18.5	110.9	163.9
N = 4	max	467	745	918	267	396	514
	mean	54.4	153.8	219.5	24.4	126.0	182.1
	total [in M]	36.5	103.3	147.4	22.5	116.1	167.8
N = 5	max	525	710	851	290	431	549
	mean	61.6	168.3	236.9	31.7	124.3	183.6
	total [in M]	41.2	112.5	158.3	29.2	114.6	169.2

Table 3.12: The influence of the traversed voxel length per ray on the performance is shown for the parameters next to the performance drop in Fig. 3.34. The highest iteration count per column is marked in orange while the lowest iteration count per column is marked in blue.

1e-5 is used for the shortest distance while 1e-2 is used for the longest distance. Fig. 3.34 shows the measured results for varying N.

Choice of optimal N is independent of ray length

If both scenes are compared, it can be seen that a constant N is not optimal. While N = 4 performs best for all ray lengths of Hairball 3, N = 3 has the best performance for all ray lengths of Sponza 2. The diagram shows that a varying ray length is not influencing the optimal choice of N. Except for N = 5 between first hit and smallest factor, the ranking of the N values is constant for all ray lengths. For the drop of N = 5 below the performance of N = 2, the negative influence of high register pressure and memory access latency becomes visible again (cf. Sec. 3.5.6.1). Since the memory consumption per thread is already at maximum for N = 5, maintaining additional information of the ray length leads to a register spilling which results in a performance drop. Since this behavior is independent of scene complexity, this effect is perceived in both scenes.

Iteration count is increased when exceeding the ray length factor 1e-3

A performance drop is encountered if the ray length factor gets 5.5e-3. Therefore, Tab. 3.12 provides a more detailed overview. Between the factors 1e-3 and 5.5e-3, the iterations triple for Hairball while they increase by an order of magnitude for Sponza. The performance drops in similar ratios. It is assumed that the iteration count increases due to the scene complexity. Both scenes are based on surface meshes with the minimum surface thickness defined by one voxel.

Therefore, the distance between surfaces leads to a higher iteration count as well. In the case of the spatially sparser Sponza scene, rays pass the first surface and need to traverse the scene with a large number of iterations until the next surface is encountered. Therefore, the drop seems to be more abrupt than in the Hairball scene which has more surfaces next to each other and less empty space needs to be skipped during the traversal.

3.6 SUMMARY

Main goal of the proposed SVT is an improvement of THz simulations

With the goal of improving THz simulations, the concept of SVTs has been proposed in this chapter. Based on the ideas for very efficient rendering of SVOs of [LK10] and the memory-efficient processing of sparse volumes of [CNLE09], SVTs have been developed to achieve a processing of thick surfaces which allows to incorporate physical properties of THz radiation. The main idea of SVTs is the generalization of SVOs of [LK10] to increase the number of children per node which allows scene representations at a higher resolution. A voxelization and a rendering have been developed for processing SVTs.

Proposed GPU voxelization works out-of-core and grid-free

The voxelization uses triangle meshes as input and creates SVTs for a varying N as output, i.e. the maximum number of children per node can be chosen. The processing is accelerated by the GPU to allow a very performant SVT creation. A streaming concept has been proposed. It allows to process only subsets of triangles and nodes of the SVT. Therefore, the implemented method is out-of-core. Additionally, the processing is grid-free which results in a performance gain and a reduced memory consumption in the process of creating SVTs.

The stack-based SVT rendering allows different applications

A raycasting approach is used for rendering the created SVTs with a varying brick size depending on N. The rendering approach is stack-based, i.e. the state of the traversal is stored on every hierarchy level if the level needs to be switched. Efficient accessing of child nodes by single pointers and bitmasks leads to a performant ray processing. Depending on the application, the traversal is terminated after the first surface hit or continues until the ray leaves the scene representation completely. Furthermore, the ray can be terminated if it fulfills different criteria like the passed distance inside voxels, which can be interpreted as penetration depth in a THz simulation.

Beam optimization of [LK10] is corrected

While [LK10] focuses on the optimizations for surface rendering, the SVT rendering is kept more general and can not use specific surface optimizations like the contour slabs. Another optimization of [LK10] is beam optimization which saves performance by starting the ray traversal nearer to the final ray termination. This optimization can be used for SVTs as well. Here, an improved dilation step is introduced to remove visual artifacts that occur in the original variant of [LK10].

Choice of optimal N depends on complexity and sparsity of the scene

The choice of N has the largest influence on the behavior of SVTs. It influences the memory consumption and the rendering performance. For the memory consumption it is shown that N = 5 and N = 6 are optimal for voxelized surface models. If the sparsity of the scene decreases, a larger N gets optimal. For the rendering performance, N = 4 and N = 3 turned out to be the best choice in most cases. Therefore, the optimal N is not constant and changes depending on the scene properties.

SVTs serve as a solid foundation for a THz simulation

While the proposed voxelization even outperforms current out-of-core approaches for creating sparse voxel representations, it turned out that the rendering concept of the voxel structures cannot compete with specialized CG methods for common surface or volume rendering. It becomes apparent that a single thread requires more memory resources with an increasing N, because more data inside a brick needs to be maintained in the stack and more general computations are required due to the varying number of children. Furthermore, the memory consumption is reduced and a scene representation at a higher resolution can be achieved, but the desired memory savings are lower than expected, because the most memory is spent on attributes which are constant under a varying N and only the hierarchical information can be reduced. A recent work of [DKB*16] shows that a palette-based approach for representing material attributes leads to a more substantial reduction of memory requirements if the scene consists of many homogeneous color attributes. Nonetheless, SVTs provide a solid foundation for the application in a THz simulation which requires high resolution datasets that are not obtainable with usual volume approaches and a performant rendering which can consider further effects related to surface thickness or roughness.

4

SIMULATION OF WAVE EFFECTS

The topic of this chapter is the simulation of wave effects and the application of the proposed SVT (see Chap. 3) to such a simulation. The focus lies on those physical effects, which influence the THz propagation and extend simulation models of ray optics. After a motivation (see Sec. 4.1), a general overview of methods for simulating electromagnetic waves is given in Sec. 4.2. Related work on THz simulations is discussed in Sec. 4.3. Sec. 4.4 provides an introduction to physical wave effects which have the largest influence on the imaging. To consider these effects, two simulations with different focuses have been implemented. They are discussed in Sec. 4.5. While the goal of the first simulation is the simulation of a hybrid THz scanner which applies a synthetic aperture image reconstruction (see Sec. 5.2), the second simulation uses the SVT to allow a THz simulation with multi-layered materials in a simpler THz scanner setup (see Sec. 5.3). These simulations are discussed in Sec. 4.5.1 and Sec. 4.5.2, respectively. A summary with a discussion of limitations and future work is given in Sec. 4.6.

4.1 MOTIVATION

Advantages of THz simulation

The simulation of electromagnetic radiation is an essential technique for applications in the fields of wireless communication, remote sensing or radar. Simulation methods allow to evaluate the quality of imaging systems even if they are not built yet. These approaches get more important if the equipment has very high production costs or long development phases which is usually the case for prototypical setups. Specially, THz hardware is very expensive currently. Therefore, the development process for THz imaging systems gets cheaper if a reliable and efficient simulation is available.

SVT provides precise and performant results

While the efficiency of the THz simulation can greatly be improved by exploiting the GPU, the use of a data structure like the SVT (see Sec. 3.3) is beneficial for a THz simulation in terms of precision and reliability. This structure allows to store object representations in a high resolution while keeping the memory consumption low. Furthermore, a fast ray traversal of this structure (see Sec. 3.5.4) ensures either a performant processing or an even more precise evaluation of scattering behavior.

4.2 OVERVIEW OF SIMULATION METHODS

Numerical approaches are used for real-world scenarios

The complexity of electromagnetic radiation in real-world scenes does usually not allow a simulation with analytic methods for larger scenarios. Therefore, electromagnetic fields are modeled by numerical approaches. Those approaches can be categorized into exact and asymptotic methods.

Overview of exact methods

The basic idea of exact methods is the spatial discretization of the scene to a uniform grid. For each grid cell, Maxwell's equations need to be solved. In the case of local methods, differential equations are used to calculate values at the grid cells throughout the scene directly. Examples for this local approach are finite element-, finite difference-, or finite volume methods. In global methods, boundary conditions and integral equations are used to calculate the solution for each cell in the grid. Methods of moments and methods of lines are two examples for these global methods. Further information on those methods of computational electromagnetics (CEM) can be found in [RBI12, CJMS00, Dav10].

Overview of asymptotic methods

As opposed to exact methods, asymptotic methods do not need a uniform discretization of the scene. Therefore, they do not represent empty space. Instead, surface representations of outer geometry and a concept like rays or beams are used to calculate the propagation of the electromagnetic field by high frequency approximations, which neglect or simplify the influence of wave effects. This leads to less accurate results, because these approximations lose validity in the case of rougher surfaces or lower frequencies. However, these simplifications lead to a strongly reduced computation time.

Asymptotic methods are used in this thesis

Depending on the application, it is beneficial to use exact methods, asymptotic methods or a mix of both. For simulating large scenes with a lot of empty space like it is the case for stand-off THz scanners, it is not feasible to use exact methods because available computer memory limits the possible grid resolution of the simulated scene. Even if enough memory would be available, the calculation time would be very high. Therefore, this thesis focuses on using asymptotic methods to simulate THz radiation, because less memory is needed and a higher performance is achieved.

4.3 RELATED WORK

Raytracing builds the foundation for asymptotic methods

Asymptotic methods base on the fundamental principle of raytracing (see Sec. 2.1.2.2) for simulating the field propagation. In the CEM community, this principle is known as Shooting and Bouncing Rays (SBR) [LCL89]. Depending on radiation- and scene-properties, techniques of geometrical optics and physical optics can be used to calculate the field contribution.

The methods of this thesis use SBR with physical optics

Geometrical optics are useful, if the simulated frequency is high in comparison to surface structures in the scene, because they rely on high frequency simplifications. If wave effects like diffraction or interference influence the result, geometrical optics gets less accurate. Those wave effects can only be simulated by physical optics, because here the interaction between rays is considered by integrating contributions over a surface. For the application of a THz simulation, wave effects play an important role for the image formation process, because the frequencies and surface structures of real-world scenes have similar dimensions. Hence, the developed methods in this thesis can be classified as an SBR method using physical optics.

First methods simulate passive THz systems only

First simulations of millimeter waves or terahertz radiation for the use in security systems ([GGLH06a]) or wireless communications ([PJJK11]) appeared. Those approaches use only passive radiation for the simulated imaging. They are not applicable to systems which use active radiation with coherent computation (cf. Sec. 2.2.2), because they do not incorporate a phase consistent surface reflectance model.

Current methods incorporate the simulation of incoherent radiation

The Helmholtz-Kirchhoff integral is the basis for the simulation of active radiation. This integral is used in the Beckmann-Spizzichino model to describe the scattering of the electromagnetic field for specular reflections. [NIK91] provides a good explanation of this model. For example, [PJJK11] assigns this model to the scattering of non-specular surfaces. This Kirchhoff approximation allows only incoherent computation, but the simulated systems in this thesis rely on coherent computations for imaging.

[CV07] simulates coherent radiation and active systems

Therefore, a simulation of phase information is required. [CV07] extends the Kirchhoff approximation to the simulation of coherent radiation by calculating complex electric amplitudes. The proposed extensions in this thesis base on the method of [CV07].

4.4 MAIN INFLUENCES ON A THZ SIMULATION

Short overview of additional influences on THz simulation

The following section gives a rough overview of the physical properties which need to be considered for simulating THz radiation in addition to the basic CG techniques of local illumination (see Sec. 2.1.3.1). While this overview focuses on short explanations and definitions on the most significant influences, [Hec02] and [ST07] are recommended for more in-depth information on these physical properties.

4.4.1 *Wave Properties*

Principles of wave optics need to be considered

The scattering of electromagnetic radiation can only be simulated by geometrical optics if the wavelength of the radiation is much smaller than the irradiated surface structures, because principles of wave optics have a negligible influence on the image formation process in this case. Since wavelengths of THz radiation and real-world surface

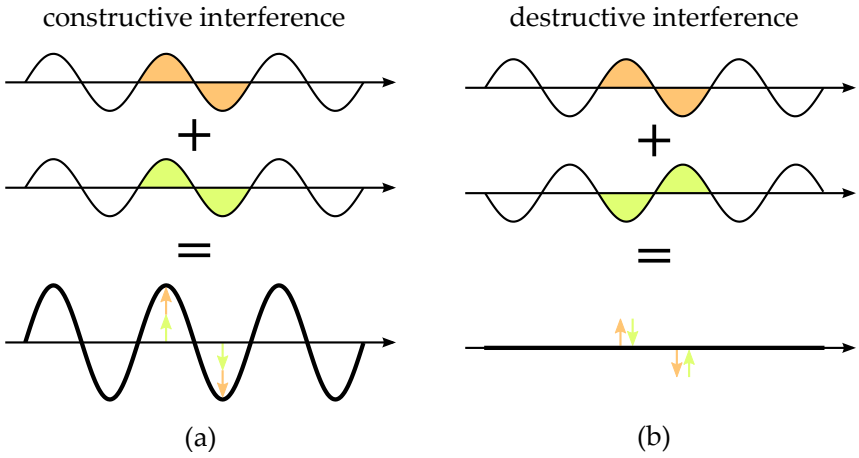


Figure 4.1: The principle of interference is shown for two waves with the same frequencies. If the waves are in phase, the summed signal is amplified and constructive interference is the result (a). If the waves have a phase difference of π, the signals are canceled out and the result is a destructive interference (b).

structures both lie in the sub-millimeter range ($10^{-}4$m - $10^{-}3$m), the ray approximation of geometrical optics gets incomplete, so that wave properties need to be considered in a simulation of THz imaging.

Superposition of waves

These wave properties base on the principle of superposition. It states that waves superimpose if they share the same geometrical position at the same time without influencing the original properties of the waves. If the waves passed each other, the waves get back to their unaltered state before the interaction.

Interference is a result of superposition

Interference is the direct result of this superposition of waves. Depending on phases, amplitudes and wavelengths of the interacting waves, a constructive or a destructive interference is the result. While

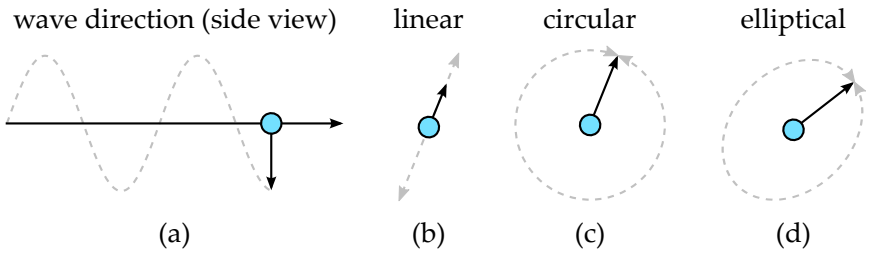


Figure 4.2: In a plane which is perpendicular to the wave direction, the movement of a vector can be used to describe the polarization of the electric field over time (a). A scaling of the vector represents a linear polarization (b). If the vector rotates without scaling, the polarization is circular (c). The most general case is the elliptical polarization, where the strength of the electric field varies over time (d). The movement of the vector can be clockwise or counterclockwise.

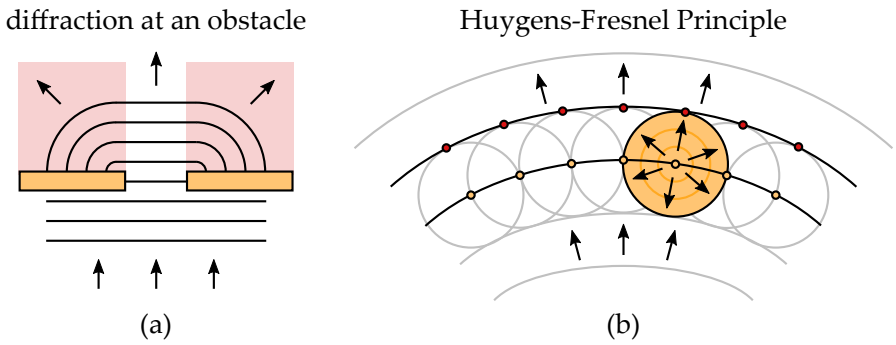


Figure 4.3: Depending on obstacle size and wavelength, diffraction is occurring. Here, the wave bends around the obstacle and propagates into shadow regions (a, marked in red). The Huygens-Fresnel Principle states that the wavefront of a primary wave (red dots) can be created by the superposition of spherical waves which were placed on the wavefront (orange dots) at an earlier point in time (b).

a constructive interference leads to an amplification of the signal, a destructive interference leads to a reduction or even a complete vanishing of the signal. Depending on the phase difference, this interference varies between these two cases. If the superposed waves have different frequencies, the interference varies along the wave propagation. Fig. 4.1 shows an example of both types of interference for a simplified case with equal frequencies and amplitudes.

Polarization describes the oscillation behavior

The oscillation behavior of the electric field of a wave is described by polarization. If the wave encounters an object, the polarization may change. It follows that the polarization is not fixed during the propagation. To describe such a behavior, the oscillation of the wave is represented by a transformation of a vector which lies in a perpendicular plane to the propagation direction. The general case is the elliptical movement of this vector inside the plane. If the strength of the electric field is constant while the wave propagates, the polarization is circular and the vector rotates around a circle. If the movement of the electric field is limited to a scaling of the vector, the polarization is linear. Fig. 4.2 depicts the variants.

4.4.2 *Object Interaction*

Diffraction

Diffraction is another property of wave optics. In comparison to interference and polarization, an interaction of the wave with an obstacle is required. It states that the wave propagates around corners or along the boundary of this obstacle, so that the wave reaches shadow regions (see Fig. 4.3 (a)). The interference patterns of a diffraction can be approximated by the Huygens-Fresnel Principle.

Formula	Description
$R_a = \frac{1}{n}\sum_{i=1}^{n} \lvert y_i \rvert$	Arithmetic mean
$R_q = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2}$	Root mean square
$R_v = \min y_i$	Maximum valley depth
$R_p = \max y_i$	Maximum peak height
$R_t = R_p + R_v$	Maximum height of profile
$R_{sk} = \frac{1}{nR_q^3}\sum_{i=1}^{n} y_i^3$	Skewness (symmetry)
$R_{ku} = \frac{1}{nR_q^4}\sum_{i=1}^{n} y_i^4$	Kurtosis (sharpness)

Table 4.1: Common parameters for describing the surface roughness are provided (see [ISO10] for further details). y_i is the i^{th} difference of y between the mean line of a one-dimensional surface profile and the measured surface point. The mean line of the profile is set to the height which minimizes the height deviations.

Huygens-Fresnel Principle

The Huygens-Fresnel Principle states that a wavefront can be replaced by a finite number of spherical waves with the same frequency, so that the superposition of these waves leads to the same envelope which would be created by the original wavefront at a later point in time. Fig. 4.3 (b) visualizes this behavior.

Scattering at rough surfaces is an active research area

In case of a THz simulation, diffraction influences the current approximations of geometrical optics for calculation of reflection and transmittance. Since surface irregularities have a comparable size to the wavelength, a wave can not be approximated by a ray. The diffraction behavior under these conditions is still an active field of research since common scattering approximations assume a wave interaction with smooth surfaces.

Roughness is described by statistical models usually

A roughness representation for surfaces is required as well. Therefore, either highly detailed object representations or statistical models are needed to describe the roughness. Commonly, statistical models are used to define the roughness implicitly and calculated properties like surface normal or traveled distance of a ray are adjusted by the generated roughness values. Tab. 4.1 gives an overview of common parameters for describing surface roughness statistically.

Signal attenuation by transmittance needs to be considered

Another important aspect of the object interaction in the THz range is the simulation of signal attenuation inside the radiated material, because many real-world materials are partly invisible in the THz range and the wave propagation continues after traversing the material. This leads to the need that object representations have inner structures. The extinction coefficient of a complex index of refraction can be used to compute the absorption of the material, but the traveled distance of the radiation inside the material must be known. Illumination techniques for volume rendering consider distance-dependent

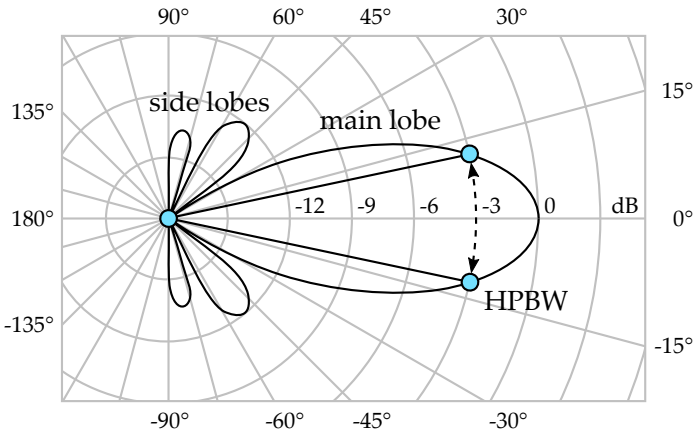


Figure 4.4: An exemplary radiation pattern shows the directivity curve of the antenna. The main lobe with the strongest power density is centered around the main direction at 0°, while the side lobes have a much weaker power density next to the main lobe. The half power beamwidth (HPBW) is an angle to describe the region where the power density drops to -3 dB (dashed line between the blue dots).

attenuation effects already. These effects are based on calculations with intensity. Since a simulation of coherent radiation is required, a calculation with phases needs to be considered. In the case of an illumination of surface meshes, physical effects which depend on the penetration depth are only approximated by statistical methods.

4.4.3 *Acquisition System*

Simulation of antennas and a lens system are needed

Other large factors for a THz simulation are the creation of the radiation and the properties of the beam. In this thesis, it is assumed that antennas are used for receiving and transmitting THz signals (see Sec. 2.2.2). Additionally, it is common that a lens system focuses the beam before the interaction with the scene object to optimize the imaging quality.

Directivity provides the distribution of power density

While the complete electromagnetic simulation of inner antenna structures is out of scope for a THz scattering simulation, the property of the outgoing antenna radiation is required. Therefore, the directivity of the antenna needs to be simulated. A radiation pattern shows the distribution of the radiation into individual angle-dependent directions. Usually it consists of a main lobe in the main direction of the antenna which holds the highest power density. Next to the main lobe, additional side lobes with lower power density are present. An important property is the half power beamwidth, which defines the opening angle of the antenna, where the signal reduces to -3 dB in comparison to the signal of the main direction. Fig. 4.4

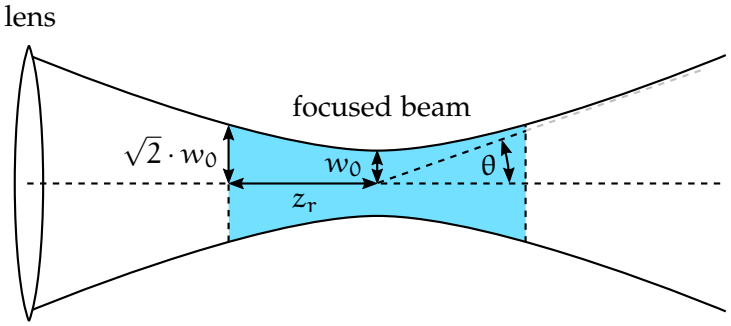


Figure 4.5: The beam parameter product $w_o \cdot \theta$ and the Rayleigh length z_r of a focused beam are illustrated. z_r defines the depth of focus, which represents the region around the focus point where the object can be acquired with an acceptable image quality. w_o provides information about the spatial resolution of the system, i.e. the minimal size of detectable features.

illustrates the mentioned terms. To simulate the propagation of the radiation, this curve of the antenna must be sampled.

Influences on imaging quality

If a lens system for focusing is used, additional principles of optics become important. In the case of THz radiation, the beam parameter product and the Rayleigh length are important for simulating a more correct imaging. The beam parameter product provides the information of the beam quality, which is linked to the image quality directly, because it determines the spatial resolution of the system. It is given by $w_0 \cdot \theta$, where w_0 is the beam waist or radius of the focus point and θ is the opening angle in the far field (see Fig. 4.5). The Rayleigh length z_r describes a similar behavior along the radiation direction, because it defines the depth of focus, which is the region before and behind the focus point which can be resolved with an acceptable image quality (see blue region in Fig. 4.5). At the end of both sides, the focus point size increased to its double. z_r is calculated by $\frac{\pi \cdot w_o^2}{\lambda}$.

4.5 THZ SIMULATIONS

[Kli12] focuses on the scanner of Sec. 5.2

In the following section, two approaches for simulating THz radiation are presented. In the first approach of [Kli12] (see Sec. 4.5.1), a specific simulation for the hybrid scanner of Sec. 5.2 has been developed. In terms of CG expressions, this simulation represents a local illumination which is extended by physical effects of wave optics.

SVT simulation

In the second approach (see Sec. 4.5.2), the SVT of Chap. 3 is used to extend a THz simulation by inherent features of the proposed voxel data structure. It features highly detailed surface representations and the calculation of precise ray distances for simulating material absorption. The simulation can be seen as a global illumination approach in the THz range, because the scattering is not restricted to a single

bounce at the first object. Nonetheless, it is limited to a subset of possible radiation paths and physical effects which can be simulated.

Main differences between [Kli12] and SVT simulation

The main differences between both approaches lie in the scene representation and the ray traversal. While a statistical ray sampling with densities below $\lambda/10$ for the simulated wavelength is used in [Kli12], the SVT rendering samples explicit geometry defined by voxels. Furthermore, the simulated systems differ. While the purpose of [Kli12] is a simulation of unfocused radiation for synthetic aperture imaging, the SVT rendering is used to simulate focused radiation of transmitters and receivers for THz scanning approaches. Tab. 4.2 provides more detailed information on the comparison between the two approaches with respect to material interaction, scattering behavior and radiation properties.

Combining both approaches is not trivial

From the comparison can be seen that the complexity of different physical effects varies for the two implementations, e.g. the correct handling of the polarization states after reflection get more complicated in a multi-bounce approach. Therefore, a combination to one unified solution is not trivial due to the distinct feature set.

Structure of this section

First, the implemented effects of [Kli12] and results are shown. Afterwards, the SVT simulation is presented. Here, the adjustments to the SVT methods are presented. The differences to the implemented physical effects of [Kli12] and the added features to the SVT simulation follow. This section ends with results of the simulation with SVTs.

4.5.1 *Hybrid Setup Simulation*

Overview

The following section is based on [PKK*13], which describes the simulation of [Kli12]. The developed approach is implemented by a volume rendering which uses raycasting to simulate the wave propagation of THz radiation. The irradiated simulation objects inside the volume are obtained from a common voxelization of surface meshes, so that each voxel holds an isovalue. The determination of surfaces is done by isosurface rendering (see Sec. 2.1.2.2). After describing the basic approach of the THz simulation in Sec. 4.5.1.1, Sec. 4.5.1.2 describes the additional physical effects which are considered in the simulation. Results of the simulation are shown in Sec. 4.5.1.3.

4.5.1.1 *Basic Approach*

Properties of the ray traversal

While the simulated effects have a general applicability, the hybrid scanner of Sec. 5.2 serves as application example and the ray traversal is adapted to the corresponding configuration (see Sec. 5.2.2). Here, a particular measurement consists of several antennas which transmit THz radiation and several antennas which receive THz radiation, i.e. a multiple-input multiple-output approach (MIMO) is used. For one simulated measurement of this configuration, all rays between trans-

mitters and receivers are distributed in a 2D plane, i.e. a slice, which intersects the scene representation in a volume grid. The rays are traversed along the depth of the plane. Since the imaging is based on a reconstruction by synthetic aperture methods in the vertical dimension, unfocused radiation of each antenna is simulated by a shooting of equally distributed rays along the whole height of the 2D plane. A wavelength-dependent spacing of $\lambda/10$ between the individual rays is taken from the SBR technique of [LCL89].

Phase is calculated by frequency and ray distance

For a simplified visibility check, the rays start at the receiver and stop after the first isosurface is found at a surface position v. A ray from v to the transmitter is sent without traversing the volume again and the traversal is terminated. Therefore, additional reflections or physical effects which occur behind v have no influence on the simulation. With a frequency sample f_k and the length d_m of the complete ray between transmitter t, v and receiver r, a phase $\varphi_{k,m}$ can be calculated by Eq. 4.1. Additionally, the simulated strength of the received signal is represented by amplitude $A_{k,m}$. In the basic approach, an attenuation of the signal is not simulated so that $A_{k,m} = 1$ is applied.

$$\varphi_{k,m} = 2\pi f_k \frac{d_m}{c} \tag{4.1}$$

$$A_{tr,k} \cdot e^{-i\varphi_{tr,k}} = \sum_{m=1}^{M} A_{k,m} \cdot e^{-i\varphi_{k,m}} \tag{4.2}$$

Superposition of waves is incorporated

To calculate an overall phase $\varphi_{tr,k}$ and an overall amplitude $A_{tr,k}$ which represent the contributions of all rays M for a combination of a transmitter and a receiver tr, the individual values of each ray m are accumulated. This accumulation corresponds to the simulation of the superposition of waves and is shown in Eq. 4.2. It follows that an output for the reconstruction (see Sec. 5.2.4) consists of a matrix that holds amplitudes $A_{tr,k}$ and phases $\varphi_{tr,k}$ for each frequency sample and for each combination of a transmitter and a receiver.

4.5.1.2 *Simulated Influences*

[CV07] extends the basic simulation

Interference is the only effect of wave optics, which is considered in the basic approach. The approach needs to be extended by the additional effects of wave optics which influence the scattering of THz radiation as well (see Sec. 4.4). For the THz case that wavelength λ and standard deviation of the surface heights σ_h are similar, the Beckmann-Kirchhoff theory ([BS87]) can be used to calculate the roughness scattering. However, this theory does not provide a coherent calculation with complex electric amplitudes. Since the work of [CV07] allows such a coherent computation, it serves as basis for the implemented simulation. The approach of [CV07] holds the following three important properties:

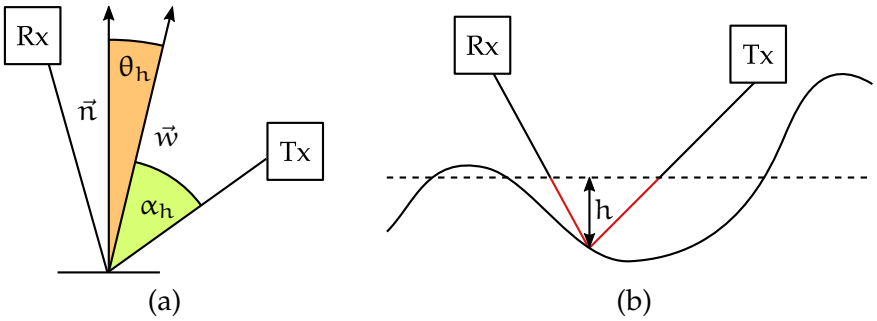


Figure 4.6: Parameters for the reflection mechanisms are depicted. α_h and θ_h are angles used for attenuation by Fresnel reflection and surface roughness (a). A random height h is used for calculating roughness by random phase offsets. These offsets are obtained by adjusting the ray distance between Rx and Tx (b, red lines). Image source: [PKK*13]

- A normal distribution function (NDF), which depends on the standard deviation of the surface heights σ_h and the correlation length L_c, is used for the ray-based scattering.
- Traveled ray distance, Fresnel reflection and surface roughness are used to calculate the phase characteristics. The surface roughness introduces random phase shifts φ_{rough} due to a phase variance function that involves σ_h.
- The coherent sum is calculated from individual ray contributions.

Diffraction of rough surface scattering

To simulate diffraction in the THz range, the influence of rough surface scattering needs to be considered. Therefore, the signal attenuation and phase offsets are simulated. While [CV07] calculates random phase shifts by micro-facets, [Kli12] uses precomputed height profiles to simulate phase offsets φ_{rough} from a rough surface scattering. After an intersection point v of ray and surface is found, the lookup in this profile gives a random height h, which is added to the traveled ray distance. Therefore, the phase of the corresponding ray is changed (see Fig. 4.6 (b))

Generation of height profiles for phase offsets

To create these height profiles, a standard deviation of heights σ_h and a correlation length L_c for describing the surface roughness need to be provided. These two parameters are used to create one-dimensional height profiles by the algorithm of [BPK07, BPK08][1]. It consists of the following steps:

1. Based on the Gaussian distribution, random heights h_u are generated with a standard deviation σ_h:

1 http://www.mysimlabs.com/surface_generation.html

$$\rho_{h_u}(x) = \frac{1}{\sqrt{2\pi}\sigma_h} e^{-\frac{x^2}{2\sigma_h^2}}$$

2. Based on the Gaussian correlation function, a filter kernel $F(x)$ is used to achieve distance related dependencies of similar surface features:

$$F(x) = e^{-\frac{x^2}{\frac{L_c^2}{2}}}$$

3. A correlated distribution of heights is generated by the convolution of $h_u(x)$ and $F(x)$:

$$h(x) = (h_u * F)(x)$$

Signal attenuation by roughness

The signal attenuation of the rough surface scattering is computed by the normal distribution function ρ_{NDF} for Gaussian rough surfaces from [CV07] in Eq. 4.3. Fig. 4.7 shows the angle-dependent attenuation of exemplary roughness parameters σ_h and L_c.

$$\rho_{NDF}(\theta_h) = \frac{L_c}{2\sqrt{\pi}\sigma_h \cos^2(\theta_h)} e^{-(\frac{\tan(\theta_h)L_c}{2\sigma_h})^2} \tag{4.3}$$

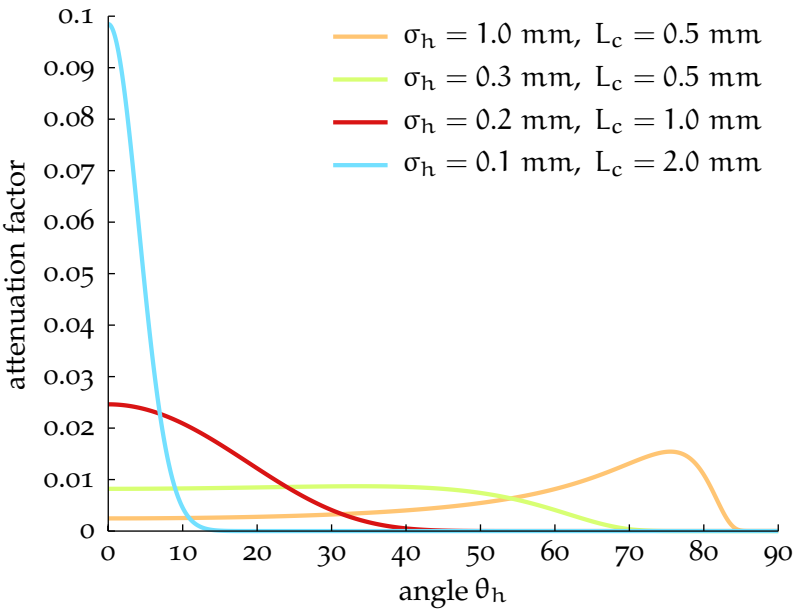


Figure 4.7: Influence of standard deviation of surface heights σ_h and correlation length L_c to the attenuation by surface roughness: Increasing σ_h and decreasing L_c lead to a rougher surface and the contribution is spread to wider angles. Image source: [PKK*13]

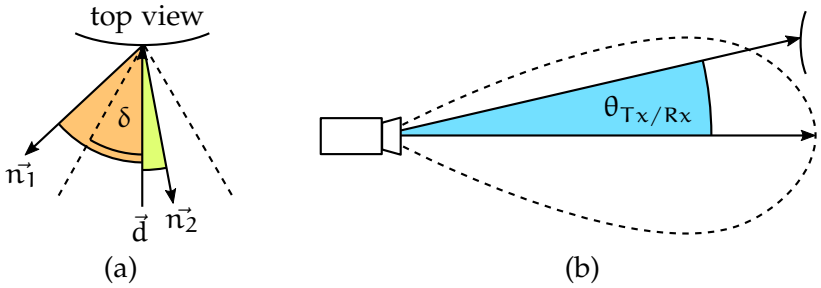


Figure 4.8: Angle threshold δ is used for taking into account normals ($\vec{n_2}$) that slightly shift from the slice direction $\vec{d}$, while normals outside this angle ($\vec{n_1}$) are neglected for further computation (a). $\theta_{Tx/Rx}$ defines the angle between main direction of the antenna and direction between antenna and intersection point on the surface for calculating an attenuation that depends on the half power beamwidth θ_{HPBW} (b). Image source: [PKK*13]

This method of rough surface scattering is combined with the computation of the Fresnel term for a complete description of the reflection. In the implemented simulation, all determined intersections have the same material because the used volume representation does not store material attributes. A mapping of isovalue ranges could be used to incorporate such a simulation of multiple materials in one simulation. The Fresnel reflection is calculated with complex indices of refraction and is always done from air into a material because the rays are sent back to receiver after the first surface intersection.

Fresnel reflection

The combined reflection mechanism is shown in Eq. 4.4 and Eq. 4.5. It incorporates the phase shifts and signal attenuation from rough surface scattering and the Fresnel reflection F. $F_{\parallel}$ represents the parallel polarization state and $F_{\perp}$ represents the perpendicular polarization, so that a final reflection vector $\vec{f} = (f_{\parallel}, f_{\perp})^T$ is created. The additional parameters in Eq. 4.4 and Eq. 4.5 are the incidence angle α_h, the position on the surface $\vec{x}$ and the deviation angle θ_h between macroscopic surface normal $\vec{n}$ and the microfacet normal $\vec{w}$. $\vec{w}$ is the halfway vector between transmitter and receiver (see Fig. 4.6 (a)).

Fresnel reflection and rough surface scattering are combined

$$f_{\parallel}(\vec{x}, \theta_h, \alpha_h) = \rho_{NDF}(\theta_h) \cdot F_{\parallel}(\alpha_h) \cdot e^{-i\varphi_{rough}(\vec{x})} \tag{4.4}$$

$$f_{\perp}(\vec{x}, \theta_h, \alpha_h) = \rho_{NDF}(\theta_h) \cdot F_{\perp}(\alpha_h) \cdot e^{-i\varphi_{rough}(\vec{x})} \tag{4.5}$$

A ray traversal in a 2D plane leads to the simulation of a perfectly focused beam, i.e. only those surface normals which lie in that plane contribute to the received signal. Since a real beam has an extent and encounters varying normal directions, a user-defined threshold δ is used to simulate a more realistic focusing without shooting additional rays. If the angular deviation between the slice direction $\vec{d}$ and the surface normal $\vec{n}$ is smaller than δ, the surface is considered for further reflection calculation as well (see Fig. 4.8 (a)).

More realistic focusing

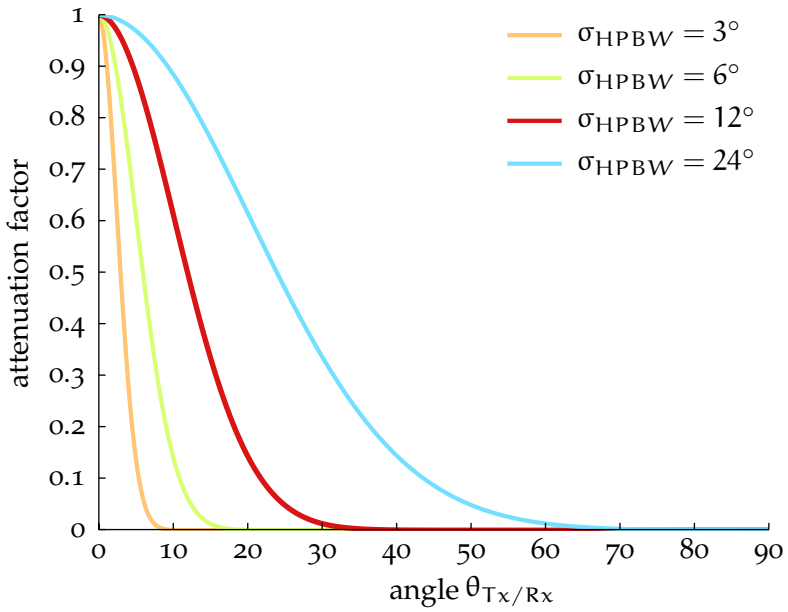


Figure 4.9: Depending on the half power beamwidth θ_{HPBW}, the attenuation factor is enlarged to wider angles $\theta_{Tx/Rx}$. Image source: [PKK*13]

Antenna properties

Horn antennas serve as transmitters and receivers. They are modeled as isotropic point sources with a main direction, i.e. side lobs of the antenna are neglected. An angular deviation $\theta_{Tx/Rx}$ between main direction and outgoing ray direction is compared with the angle of the half power beamwidth of the antenna θ_{HPBW} and determines the drop in magnitude of the transmitted or received radiation (see Fig. 4.8 (b)). Fig. 4.9 shows the influence of exemplary values for θ_{HPBW}. The used distribution function (see Eq. 4.6) is based on the antenna model of [TSI*06]. Here, the part for calculating the attenuation of the main lobe is used.

$$\rho_{Tx/Rx}(\theta_{Tx/Rx}, \theta_{HPBW}) = e^{-\frac{4\ln(2)}{\theta^2_{HPBW}}\theta^2_{Tx/Rx}} \tag{4.6}$$

Polarization

Frenet frames are used to rotate between antenna and surface coordinate systems to calculate polarization states. This principle of geometrical depolarization is applied in [PJK11] and [MMS*11] as well. Each ray determines its polarization state of the plane wave in its individual coordinate system. Therefore, rotation matrices $R_{\vartheta_{Tx}}$ and $R_{\vartheta_{Rx}}$ are applied before and after reflection $\vec{f}$ (see Eq. 4.4 and 4.5). To simulate linearly polarized antennas, a coefficient vector $\vec{c}$, removes the parallel or the perpendicular component of the polarization state.

Calculation per ray

The complete contribution of a ray is shown in Eq. 4.7. It is calculated by combining the reflection mechanism with polarization cal-

culation and antenna attenuation. While $\vec{E}_i$ represents the emitted plane wave, $\vec{E}_o$ represents the received plane wave.

$$\vec{E}_o = \vec{c} \cdot \rho_{Rx} \cdot R_{\vartheta_{Rx}} \cdot \vec{f} \cdot R_{\vartheta_{Tx}} \cdot \rho_{Tx} \cdot \vec{E}_i \tag{4.7}$$

Total contribution between Tx and Rx

The total contribution of a transmitter-receiver-combination is calculated by Eq. 4.8. Depending on frequency samples f, individual contributions per ray are accumulated, if the intersected surface is visible from receiver and transmitter. Here, the visibility terms $V_{\vec{Tx}\to\vec{S}}$ and $V_{\vec{S}\to\vec{Rx}}$ are used to represent the ability if a ray is able to travel to and from a surface element $\vec{S}$. M_{rays} is the total number of generated rays from each transmitter and is used as a normalization parameter.

$$\vec{E}_{\vec{Tx}\to\vec{Rx}}(f) = \frac{1}{M_{rays}} \cdot \sum_S \vec{E_o}(f) \cdot V_{\vec{Tx}\to\vec{S}} \cdot V_{\vec{S}\to\vec{Rx}} \tag{4.8}$$

4.5.1.3 *Results*

Overview

The simulation is evaluated by an analysis of parameters for the general physical effects and of specific parameters for the application in the hybrid setup of Sec. 5.2. Furthermore, a comparison with a measured dataset is discussed. To obtain the visual interpretation of the simulated values, the reconstruction of Sec. 5.2.4, the fusion of Sec. 5.2.5.2 and the visualization of Sec. 5.2.5.3 are applied.

Scene description

The simulated scene consists of a 3D model of a human, which is voxelized into a grid of 200x400x200 voxels. The measurements of [Ala04] for dry human skin at 30° are used. Here, the complex refractive indices are calculated from the relative permittivity. The provided frequency range between 76 and 100 GHz is smaller than the frequency range of the hybrid body scanner, so that the refraction indices are extrapolated as constant values. These values are taken for the whole model and all examples. The only exception is the comparison with a measurement in the last paragraph of this Sec. 4.5.1.3. Here, a plastic material with an estimated index of refraction of $1.54 - i0.001$ and metal as perfect reflector are used for a better comparability.

Roughness influences the angle-dependent intensity distribution

INFLUENCE OF SCATTERING PARAMETERS The image quality is largely influenced by the simulated scattering in the THz-range. To achieve the expected surface roughness of the scene object, the roughness parameters need to be adjusted. Smooth surfaces are obtained by a high correlation length L_c and a low standard deviation of heights σ_h. A decreasing L_c and an increasing σ_h lead to rougher surfaces. The influence of the two parameters is shown in Fig. 4.10. It can be seen that a uniform intensity is achieved in Fig. 4.10 (b) while the intensity differs in Fig. 4.10 (a) and (c). In comparison to the attenuation factors of Fig. 4.7, it can be seen that L_c = 0.5 mm and σ_h = 0.3 mm lead to an almost uniform spread of intensity over the angles of

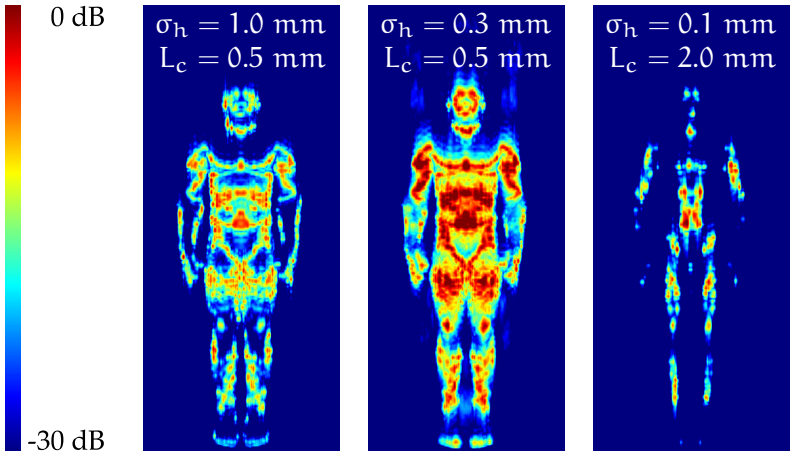


Figure 4.10: The visibility of the object in the reconstructed image is influenced by the roughness parameters. The roughness decreases from a to c: σ_h = 1.0 mm, L_c = 0.5 mm (a), σ_h = 0.3 mm, L_c = 0.5 mm (b), σ_h = 0.1 mm, L_c = 2.0 mm (c). Image source: [PKK*13]

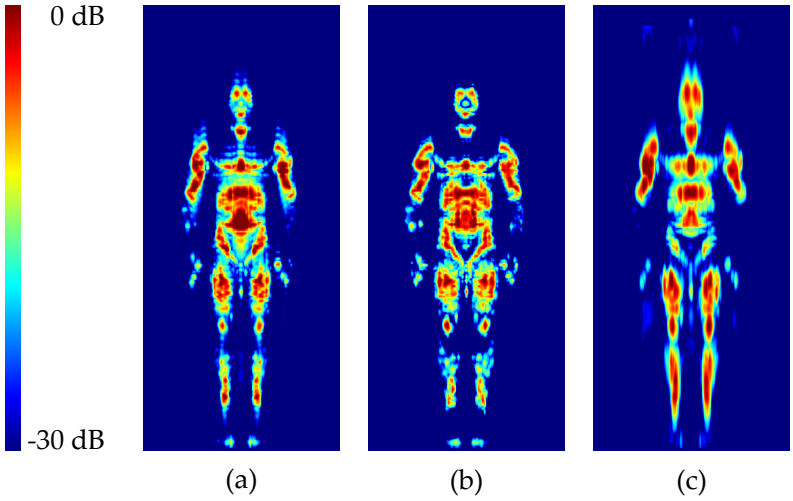


Figure 4.11: The image quality is influenced by the geometrical positioning of the setup components. All images are simulated with HPBW = 12°, σ_h = 0.2 mm and L_c = 1.0 mm. The distance between object and tr-array is 9 m (a, b), but outer mirrors are moved outwards to increase the acquisition angles (b). A sharper image and more reconstructed surfaces are obtained by a decreased distance of 5 m between tr-array and scene object (c). Image source: [PKK*13]

reflection. The roughest surface of this evaluation (σ_h = 1.0 mm, L_c = 0.5 mm, cf. Fig. 4.10 (a)) has a peak at high angles of reflections while the smoothest surface (σ_h = 0.1 mm, L_c = 2.0 mm, cf. Fig. 4.10 (c)) has a peak at low angles of reflection. It follows that the intensity

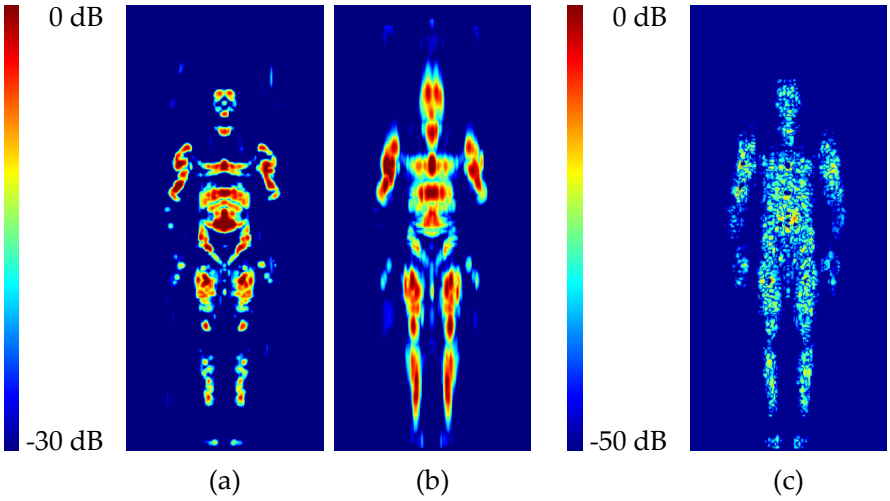


Figure 4.12: The behavior of the setup can be tested by different parameters. All images are simulated with σ_h = 0.2 mm and L_c = 1.0 mm (cf. Fig. 4.11 (a)). Transmitters and receivers are equally spaced on the linear array, so that some surfaces are not reconstructed (a). Decreasing the HPBW to 3° leads to a blurring in the vertical direction (b). A difference image between δ = 1° and δ = 10° shows the additionally acquired intensity of surfaces that deviate from the slice direction for δ = 10° (c). Image source: [PKK*13]

range of the reconstructed image depends on the range of attenuation factors over the different angles of reflection.

Geometrical system properties can be evaluated

INFLUENCE OF SETUP PARAMETERS The simulation allows to evaluate different configurations of the hybrid synthetic aperture THz system which is discussed in Sec. 5.2 and shown in Fig. 5.1. The positions of the scanner parts can be varied to test the geometrical influence on the image generation. Fig. 4.11 (a) and (b) are generated with the same simulation properties but the outer viewing mirrors of the setup are moved outwards in Fig. 4.11 (b). It can be seen that more surfaces are detected because the acquisition directions to the object differ more. If the mirror positions are not changed but the object gets nearer to the tr-array, the image gets sharper and more surfaces are detected as well (cf. Fig. 4.11 (a) and (c)). An unfocusing in the near distance is neglected because the actual beam width is always assumed to be minimal along the complete ray.

Simplified antenna properties can be adjusted

To evaluate the reconstruction model, the positions of transmitters and receivers on the linear array can be varied arbitrarily. Fig. 4.12 (a) shows the simulated radiation of equally spaced transmitters and receivers over the whole linear array. All other images of this section are generated with densely spaced receivers on the top and bottom of the linear array, while the transmitters are equally spaced in the middle of the array. The implemented spotlight model of the antenna

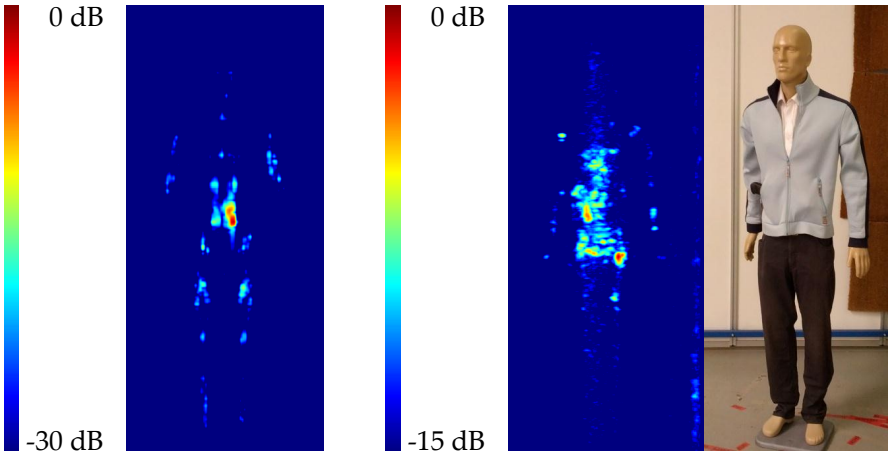


Figure 4.13: The reconstructions of a simulation and a measurement are compared. A human model with a constant complex refractive index of $1.54 - i0.001$ for a plastic material is simulated, while the right side of the stomach is simulated as metal. The used HPBW is 12° and the roughness values are σ_h = 0.1 mm and L_c = 2.0 mm. The normalization of all intensities with the maximum (metal) lowers image contrast (a). A measurement of a mannequin (c) with noise, multi-reflection and background-scattering can be seen. A weapon in the left part of the jacket and some metal in the right pocket of the pants show the highest intensity (b). In comparison to the simulated image, more scattering due to multipath effects is visible while the specular reflection looks similar. Image source: [PKK*13]

allows the evaluation of a varying HPBW. In comparison to Fig. 4.12 (a) with a HPBW of 12°, Fig. 4.12 (b) shows that a smaller HPBW with 3° leads to a more blurred image in the vertical direction. Depending on the user-defined threshold δ for a more realistic focusing (cf. Fig. 4.8 and Sec. 4.5.1.2), the blurring in the mechanical scanning direction can be approximated. The reconstructed images of $\delta = 1°$ and $\delta = 10°$ are compared in Fig. 4.12 (c). The resulting difference image shows a gain of intensity for $\delta = 10°$ because the contributing surfaces can be oriented further away from the slice direction.

Specular reflection appears similarly

COMPARISON OF SIMULATED AND MEASURED DATA Results of the simulation and a measurement are not directly comparable, because physical properties can only be simulated up to a certain point. Therefore, a discussion on similarities is following. Regarding specular reflections, the used simulation behaves like the real measurement (cf. Fig. 4.13 (a) and (b)). Specular directions that point directly in the slice direction have a high intensity while neighboring directions with a small deviation to the slice direction are not reconstructed.

Besides the effect, that the amount of diffuse reflection is much lower in the THz regime compared to the visual light, the acquisition method of the scanner prototype leads to further signal imperfections due to, e.g., cross-talk. While the simulation can model diffuse reflection, the simulation of radiation paths is restricted to single bounces at the scene object. Therefore, the diffuse scattering due to multi-reflection between clothes (see Fig. 4.13 (b)) can not be simulated. Furthermore, noise and background-scattering are neglected as well.

Diffuse reflection is not shown

4.5.2 *THz Simulation with SVTs*

Overview

To show the applicability of the SVT representation to a THz simulation, the proposed methods of Chap. 3 need to be adapted. Here, the voxelization can be used directly, but the mesh representation needs to allow the creation of thick surfaces. The ray traversal of the SVT rendering needs larger changes for the use as a THz simulation, because a bending of rays and a storage of traversed materials are required. First, the generation of scenes is addressed. Next, the calculation of physical effects are discussed. A comparison with the methods of [Kli12] and results follow.

4.5.2.1 *Generation of Scenes*

Voxelization of Sec. 3.4 is used

A scene representation for a THz simulation needs to provide information of the inner structure of the objects, because THz radiation penetrates most of the materials and the resulting signal is influenced by the traveled distance through these materials. Since the voxelization method of Sec. 3.4 is focusing on surfaces, the thickness property needs to be provided by individual triangles.

Thick surfaces are created by consecutive triangles

While one option would be the voxelization of triangles with an additional thickness parameter, the neighboring triangles need to be considered for a closed surface representation (see Fig. 4.14 (a)). The proposed voxelization would not work out-of-core, if the triangles need to have information of neighboring triangles. Therefore, additional triangles are required to create thick surfaces from consecutive voxels. To determine the position of the triangles, it is necessary to compute the edge length e of a single voxel from the scene dimension $\mathtt{dim}$ and the voxel resolution $\mathtt{res}$ of the scene by $e = \frac{\mathtt{dim}}{\mathtt{res}}$. It follows that the maximum distance between triangles needs to be below e to create closed voxel regions without gaps (see Fig. 4.14 (b)).

Quantization error of the voxelization

Additionally, the edge length e of a voxel is linked to the quantization error of the scene representation. Resulting material layers with a thickness t between triangles will have a thickness in the range of $[e, t + 2 \cdot e[$ after voxelization, because a voxel is the smallest element and both neighboring voxels can slightly be touched by the triangle. Depending on the choice of N and the desired resolution, the surrounding bounding box of the scene might shift so that a con-

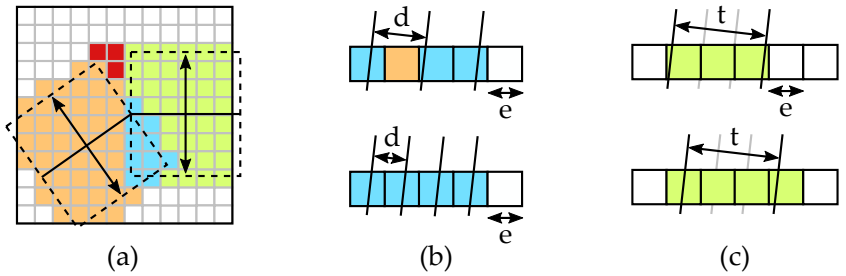


Figure 4.14: If thickness attributes at triangles are used, information of neighboring triangles is required to get a closed surface. In the given example, the red voxels would not be created without this knowledge and gaps between the extruded triangles would be the result (a). If the distance d between subsequent triangles, which represent a thick surface, is higher than the edge length e of a voxel, gaps in the material layer can occur (orange voxel, b). The position of the triangles influence a quantization error, so that a constant thickness t leads to a varying number of filled voxels (c).

stant number of triangles intersects a varying number of voxels (see Fig. 4.14 (c)). If thinner surface layers or a higher precision of the surface structure are required, res needs to be increased or dim needs to be decreased.

Voxel colors reference material descriptions

The voxel attributes of the proposed SVT implementation hold color and normal values. To allow the storage of other material parameters, which are required for a THz simulation, a lookup table is used in the rendering. Here, the created voxel colors of the underlying triangle attributes serve as identifier for a mapping to a material description, so that individual color values represent different materials. In a first approach, the complex indices of refraction are stored. Obviously, this referencing allows to extend the material description by additional parameters. The combination of several triangle attributes in one voxel is done by averaging, so that color ranges define a specific material. Since the attribute creation of the proposed voxelization is independent from the order of incoming material values per voxel, a more sophisticated technique for identifying voxel attributes could be developed.

4.5.2.2 *Adaptations to the SVT Rendering*

Overview

To adapt the proposed SVT rendering of Sec. 3.5 for the application to a THz simulation, a two-step process is introduced. In the first step, the traversed material layers between transmitter and receiver are determined. The proposed SVT rendering loop (see Alg. 3.5) is adapted to combine traversed voxels with the same material to one layer and stores an array of traversed material properties for each

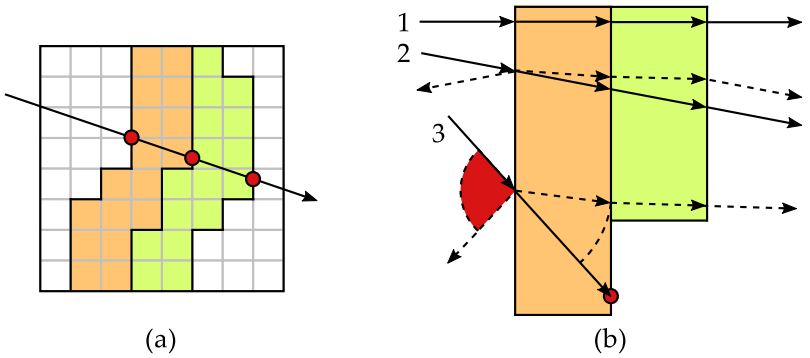


Figure 4.15: During the ray traversal, neighboring voxels with the same material are combined to one layer. The layer thicknesses are given by the red dots along the ray (a). Normal and dashed arrows are compared to show the potential error of the ray traversal without adapting the ray direction due to refraction (b). While ray 1 is not introducing an error due to a perpendicular incidence, ray 2 and ray 3 show the obtained errors from a postponed evaluation of the physical behavior for multi-bounces. While ray 2 has minor shifts in the reflection angle and the calculated layer thicknesses, the layer thicknesses of ray 3 have a large difference in the first layer and the second layer is not considered due to a shifted intersection point (red dot). Furthermore, a received reflection will be considered for ray 3 but the true reflection direction would not reach the transceiver again (red sector). The errors increase with an increasing number of material layers.

ray. In the second step, these layers are iterated to calculate the THz contribution of the respective ray for transmittance and reflection.

Creation of material layers needed for performance

ADAPTING THE RAY TRAVERSAL If the scene representation requires high voxel grid resolution and thick material layers, a large number of voxels is traversed by each ray. Since THz radiation penetrates these materials, the physical behavior needs to be evaluated at each voxel. Therefore, the performance of the ray traversal would drop significantly (cf. Fig. 3.34). To avoid this decrease of performance, a creation of material layers is proposed. Instead of evaluating the physical behavior at each voxel, neighboring voxels with the same material along the ray are identified and combined to one layer, so that the needed computations are postponed and reduced to an iteration of material layers (see Fig. 4.15 (a)).

Materials influence efficiency of layer creation

While this combination of voxels leads to a reduced computation in case of transparent THz materials, strongly reflecting materials could stop the ray traversal earlier, because no additional voxels behind this material are radiated. Hence, the materials of a scene influence the efficiency of the layer creation in comparison to an evaluation of each voxel.

Properties of a material layer

Material layers are determined during the attribute calculation of the ray traversal (see Alg. 3.5). Each material layer stores the following properties:

- traveled ray distance / material thickness
- complex index of refraction
- viewing direction / incident ray direction
- surface normal

Determination of material layers

A new material layer is created, if the material index of the current voxel is different from the material index of the previous voxel. If the material is the same, the traveled ray distance $t_{next} - t_{before}$ through the voxel is added to the total ray distance in the material. If a new layer is created, the normal of the first voxel and the current ray direction are stored. In addition, the material thickness is set to the first ray distance through the voxel. The color of the voxel serves as index for a lookup of a complex index of refraction which is stored as well.

Empty space must be interpreted as material

A special case appears, if the ray traverses empty space between full voxels. While the normal SVT rendering is skipping this space, empty space needs to be interpreted as a new material layer in the THz simulation. Therefore, the last exit point of a voxel t_{next} is stored and compared with the entry point t_{before} of the next intersected voxel. If they are not equal up to an epsilon threshold, a new layer of empty space is added before a new layer is created from the new voxel. A comparison with the previous voxel material is not needed in this case.

Unchanged ray directions constrain the applicability

Due to the complete creation of material layers before the physical behavior is evaluated, the ray direction is not adjusted by refraction mechanisms. Therefore, the layer thicknesses hold an error of shifted ray distances and intersection points after each interface (see Fig. 4.15 (b)). To correct these shift errors, every transition between two materials could be evaluated in the rendering loop, but it is not guaranteed that the resulting ray with a new direction reaches the receiver. Here, techniques like Monte Carlo path tracing can be applied to solve this problem, but the performance would decrease. Furthermore, a multi-bounce approach which tracks reflection and transmittance by additional rays at an interface would decrease the performance additionally. By contrast, the iteration of material layers allows the prediction of multiple bounces for the price of inaccuracies in the determination of layer depths. Depending on the angles of incidence of a ray, the error changes. While a perpendicular surface orientation does not introduce any error, large increasing angles of incidence lead to wrong simulation results. Therefore, the layer creation is not applicable to real-world scenarios with multi-bounce effects, but it allows to eval-

uate single bounce effects and multi-bounce reflection effects in case of perpendicular surfaces.

Beam focusing can be simulated

The performance of the SVT rendering allows to simulate further effects which influence the imaging. Since usual THz imaging systems use lens systems for focusing the radiation, the ray traversal is adapted to simulate this beam focusing and obtain more realistic results. Therefore, a sub-pixel processing is implemented. Instead of sending one ray per pixel, a varying number of rays is shot to simulate a radiation pattern and a spatial extent of the beam. Furthermore, a prototypical ray bending is implemented to simulate a more realistic focusing which is influenced by the beam parameter product and the Rayleigh length (see Sec. 4.4.3).

Each pixel represents a Gaussian beam

Each pixel holds a regular grid of rays to represent the beam. The number of rays $\#rays$ along one axis is set by a user-defined value depending on the desired precision. To simulate the radiation pattern of the beam, each ray $\vec{r}_{u,v}$ has an electric field amplitude $E_{u,v}$. This amplitude is calculated by sampling the Gaussian function to obtain the characteristics of a Gaussian beam (see Fig. 4.16 (a)). A window length $w = 5$ for a Gaussian function with $\sigma = 1$ and $\mu = 0$ is used to determine the sample distance d and sample positions x, y (see Eq. 4.9 - 4.11). $E_{u,v}$ is calculated by Eq. 4.12 and normalized by Eq. 4.13 to achieve energy conservation.

$$d = \frac{w}{\#rays - 1} \tag{4.9}$$

$$x = u \cdot d - \frac{w}{2} \tag{4.10}$$

$$y = v \cdot d - \frac{w}{2} \tag{4.11}$$

$$E_{u,v} = \frac{1}{2\pi} \cdot e^{-\frac{x^2+y^2}{2}} \tag{4.12}$$

$$\overline{E_{u,v}} = \frac{E_{u,v}}{\sum_{u=0}^{\#rays-1} \sum_{v=0}^{\#rays-1} E_{u,v}} \tag{4.13}$$

Ray bending

Since the size of a measured focus point is dependent on the wavelength and the optical lens system, linear rays can not simulate such a beam focusing (cf. Fig. 4.5). Therefore, a ray bending is used in the SVT rendering. According to the curved shape of a focused beam, each ray is separated into 4 linear segments (see Fig. 4.16 (b)). These segments are defined by lens radius r_{lens}, focal length f, Rayleigh length z_r and beam waist ω_0. For each of these segments s, a start position $\vec{o_s}$ and a normalized direction $\vec{d_s}$ of a ray are calculated (see Eq. 4.14 - 4.21). Furthermore, each ray segment stores its length l_s. During the traversal, the voxel entry point t_{before} of the current ray segment is compared to this valid ray length. If $t_{before} > l_s$, the processing of the current segment is finished, because the position on the ray is not valid. The traversal continues with the next ray segment.

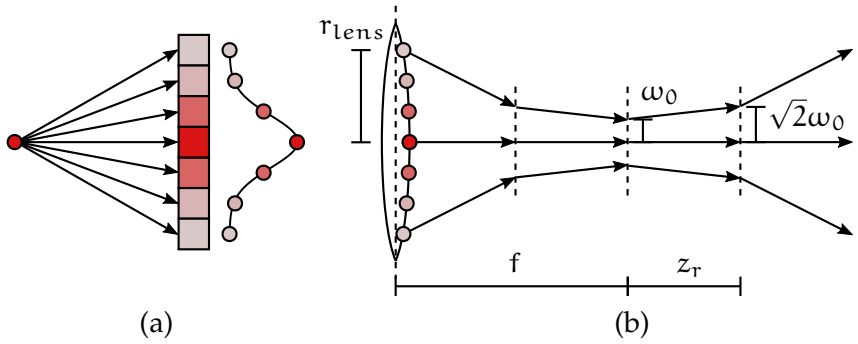


Figure 4.16: The Gaussian function is sampled by #rays in u- and v-direction to obtain the electric field amplitudes $E_{u,v}$ of the rays in one pixel. The v-direction with $\#rays = 7$ is depicted (a). Each ray consists of 4 segments to simulate a ray bending. Beam waist ω_0, Rayleigh length z_r, Lens radius r_{lens} and focus length f are used to calculate the origins and directions of the ray segments by Eq. 4.14 - 4.21 (b).

$$i = \frac{u}{\#rays - 1} - \frac{1}{2} \quad \text{with } \{u \in \mathbb{N} \mid 0 \leqslant u < \#rays\} \tag{4.14}$$

$$j = \frac{v}{\#rays - 1} - \frac{1}{2} \quad \text{with } \{v \in \mathbb{N} \mid 0 \leqslant v < \#rays\} \tag{4.15}$$

$$\vec{o_0} = (i \cdot r_{lens}, j \cdot r_{lens}, 0) \tag{4.16}$$

$$\vec{o_1} = (i \cdot \sqrt{2} \cdot \omega_0, j \cdot \sqrt{2} \cdot \omega_0, f - z_r) \tag{4.17}$$

$$\vec{o_2} = (i \cdot \omega_0, j \cdot \omega_0, f) \tag{4.18}$$

$$\vec{o_3} = (i \cdot \sqrt{2} \cdot \omega_0, j \cdot \sqrt{2} \cdot \omega_0, f + z_r) \tag{4.19}$$

$$\vec{o_4} = (i \cdot r_{lens}, j \cdot r_{lens}, 2 \cdot f) \tag{4.20}$$

$$\vec{d_s} = \frac{\vec{o_{s+1}} - \vec{o_s}}{\|\vec{o_{s+1}} - \vec{o_s}\|} \quad \text{for } s = \{0, 1, 2, 3\} \tag{4.21}$$

Calculation of layer thicknesses is changed

Due to the termination of a ray depending on its length, the calculation of layer thicknesses needs to be adapted, because a ray segment can become invalid inside a voxel. Therefore, the thickness d^{vox}_{thick} inside a voxel is calculated by Eq. 4.22. Since the starting position of the next ray segment $\vec{o_{s+1}}$ is at $\vec{o_s} + l_s \cdot \vec{d_s}$, it will hit the same voxel and continue the thickness accumulation correctly. For the special case of layer creation from empty space, the thickness calculation needs to be adapted as well, because a ray bending is not detected during the empty space skipping. Therefore, the number of processed ray segments #seg is counted and the exit point t^{last}_{next} of the last intersected voxel is stored to reconstruct the traveled ray distance $d^{\#seg}_{thick}$ by Eq. 4.23 - 4.25 when encountering the next voxel entry point t_{before}.

$$d^{vox}_{thick} = \min(t_{next}, l_s) - t_{before} \tag{4.22}$$

$$d^{0}_{thick} = t_{before} - t^{last}_{next} \tag{4.23}$$

$$d^{1}_{thick} = l_0 - t^{last}_{next} + t_{before} \tag{4.24}$$

$$d^{\#seg>1}_{thick} = l_0 - t^{last}_{next} + \sum_{s=1}^{\#seg-1} l_s + t_{before} \tag{4.25}$$

ITERATING MATERIAL LAYERS After the determination of material layers, the individual ray contributions are calculated by iterating these layers. The theoretical foundation for this calculation is based on the assumption that an inhomogeneous material volume is traversed by a ray. At each position x on the ray $\vec{r}$, an index of refraction $n(x)$ of the current material and an absorption coefficient $\alpha(x)$ are known. From $\alpha(x)$, $\kappa(x)$ can be calculated as $\kappa = \alpha \cdot \frac{c}{4\pi f}$, so that a complex index of refraction $n + i \cdot \kappa$ is available. The emitted field strength $\vec{E}_i (= \vec{E}(0))$ of a ray is used as input to calculate the outgoing electric field strength $\vec{E}_o (= \vec{E}(x))$. The material attenuation is given by Eq. 4.26.

$$\vec{E}(x) = \vec{E}(0) \cdot e^{\int_0^x -\frac{\alpha(s)}{2}\, ds} \tag{4.26}$$

Phase information is considered

To allow a complex calculation, optical path length OPL and a resulting phase φ need to be considered as well (see Eq. 4.27 and 4.28). Here, the frequency f is provided by a user-defined input value. An individual offset t_{off} per ray is used as a constant distance offset to shift φ for compensating a lens system which would provide a ray coherence in the focus point p_f, i.e. $\varphi_r(p_f) = const, \forall\, r \in rays$. If the lens system would be modeled and traversed by rays explicitly, this offset t_{off} is not required, but a less performant simulation would be the result. Extending Eq. 4.26 by phase calculations, leads to Eq. 4.29.

$$OPL(x) = t_{off} + \int_0^x n(s)\, ds \tag{4.27}$$

$$\varphi(x) = \frac{2\pi f}{c} \cdot OPL(x) \tag{4.28}$$

$$\vec{E}(x) = \vec{E}(0) \cdot e^{\int_0^x -\frac{\alpha(s)}{2} + i\frac{2\pi f}{c} OPL(s)\, ds} \tag{4.29}$$

Adding complex Fresnel calculation

To incorporate the influence of complex Fresnel terms $\vec{F}$ at the transition between material interfaces, the complex field strength is multiplied with the respective Fresnel term for transmittance $\vec{F}^t$ or reflection $\vec{F}^r$. Considering a material interface at a specific position x_0, Eq. 4.30 and 4.31 show the calculation for transmittance and reflection, respectively.

$$\vec{E}^t(x_0) = \vec{E}(x_0) \cdot \vec{F}^t(x_0) \tag{4.30}$$

$$\vec{E}^r(x_0) = \vec{E}(x_0) \cdot \vec{F}^r(x_0) \tag{4.31}$$

Homogeneous ray segments for transmittance

Since refraction occurs at discrete locations $x_1, ..., x_{N-1}$ along the whole ray, the ray is separated into N segments with lengths $\Delta_k = x_k - x_{k-1}$ for $k = 1, ..., N$, where $x_0 = 0$ is the start point and x_N is the end point of the ray. Furthermore, each segment has a refraction index n_k. Refraction takes place only at the boundaries of the material layers, so that Fresnel terms $\vec{F}_{k\to k+1}$ with $k = 1, ..., N-1$ are used. Eq. 4.32 shows the resulting calculation for transmittance. $N(x)$ is the segment in which x resides.

$$\vec{E}^t(x) = \vec{E}(0) \cdot e^{\int_0^x -\frac{\alpha(s)}{2} + i\frac{2\pi f}{c} OPL(s)\, ds} \cdot \prod_{k=1}^{N(x_k)-1} \vec{F}^t_{k\to k+1} \tag{4.32}$$

Superposition of reflected rays

The reflected signal at the transceiver is the superposition of the individual reflections for which an additional damping is considered on the way back to the transceiver. Therefore, the full reflective signal at the transceiver considering all reflections up to x_N is shown in Eq. 4.33:

$$\vec{E}^r(x) = \sum_{k=1}^{N(x_k)-1} \vec{E}^t(x_k) \cdot \vec{F}^r_{k\to k+1} \cdot ... \; e^{\int_{x_k}^0 -\frac{\alpha(s)}{2} + i\frac{2\pi f}{c} OPL(s)\, ds} \cdot \prod_{m=N(x_k)-1}^{2} \vec{F}^t_{m\to m-1} \tag{4.33}$$

Discrete variants of Eq. 4.32 and 4.33 are used

For the calculation with material layers, the discrete variants of Eq. 4.33 and 4.32 are used in Alg. 4.1. The algorithm shows the interplay between the rays of a pixel which contribute to the outgoing field strength $\vec{E}_o$. Please note that the variables are reused and results

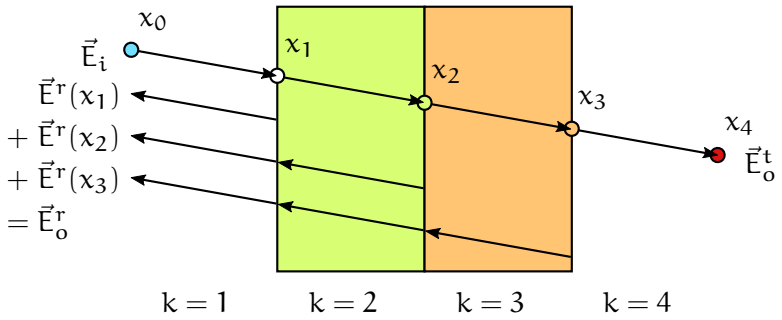


Figure 4.17: The exemplary iteration of two material layers is depicted. Starting from x_0, the interfaces x_1 - x_3 are traversed for calculating the transmitted contribution $\vec{E}^t_o$ at x_4. Furthermore, a reflected contribution $\vec{E}^r_o$ is calculated from the sum of field strengths $\vec{E}^r(x_1)$ - $\vec{E}^r(x_3)$ which are reflected back to x_0 from the respective material interfaces x_1 - x_3. The rays in the illustration are shifted away from the interface points for a better comprehensibility.

Algorithm 4.1 : Iterating the material layers from transceiver to receiver. For each material interface, the already passed layers are iterated backwards to the transceiver for calculating the reflected field strength.

```
// Final field strength of all rays in transmittance and reflection
E⃗^r_o ← 0
E⃗^t_o ← 0
for all rays of a pixel do
    // Temporary field strength per ray in transm. and refl.
    E⃗^r ← 0
    E⃗^t ← normedGauss() // Σ_{ray=1}^{#rays} E_ray = 1
    DetermineMaterialLayersByRayTraversal()
    // Iterate layers k from transceiver to receiver
    for k = 1, ..., N do
        // First layer of air is passed
        if k > 1 then
            // E⃗^r is reflected E⃗^t at interface k − 1 → k
            E⃗^r ← E⃗^t · F⃗^r_{k−1→k}
            // Iterate layers backwards to receiver after reflection
            for m = k − 1, ..., 1 do
                // Phase change and mat. attenuation in layer m
                E⃗^r ← E⃗^r · e^{−½·α_m·Δ_m} · e^{i 2πf/c · n_m·Δ_m}
                //Transm. on backward iter. without first layer
                if m > 1 then
                    //Transm. at interface m → m − 1
                    E⃗^r ← E⃗^r · F⃗^t_{m→m−1}
            // Temp. E⃗_r is added to final E⃗ in reflection
            E⃗^r_o ← E⃗^r_o + E⃗^r
            // Transmittance by Fresnel at interface k − 1 → k
            E⃗^t ← E⃗^t · F⃗^t_{k−1→k}
        // Phase change and material attenuation in layer k
        E⃗^t ← E⃗^t · e^{−½·α_k·Δ_k} · e^{i 2πf/c · n_k·Δ_k}
    // Temp. E⃗_t is added to final E⃗ in transmittance
    E⃗^t_o ← E⃗^t_o + E⃗^t
```

are accumulated in accordance with the implementation. Fig. 4.17 illustrates an exemplary traversal of two material layers with the used terminology.

4.5.2.3 *Comparison to [Kli12]*

As stated in Sec. 4.5, the simulation of [Kli12] and the THz simulation with SVTs differ strongly in the included physical effects. Tab. 4.2 provides an overview on these differences between both approaches. Similarities are restricted to the calculation of phases by a given ray

Tab. 4.2 shows differences

	[Kli12]	SVT THz Simulation
Simulated system	Unfocused radiation with synthetic aperture reconstruction, reflection	Coherent and focused lens system, reflection and transmission
Phase calculation	Given by ray distances	
Interference	Calculated by complex addition of ray contributions	
Coherency	φ = const. at antenna	φ = const. at focus point
Roughness	Statistical and constant roughness for whole scene by fixed random height profiles per 2D slice	Explicitly given by voxels
Polarization	Done by Frenet frame and Jones vector, linear polarization	Limited to perpendicular or parallel part of Fresnel equations
Occlusion between Tx and Rx	Determination of isosurfaces by rays from receiver, no occlusion check between transmitter and surface, visibility term by surface normal and antenna direction	Given by voxel materials, reflected ray directions are not altered (no occlusion possible)
Material interaction	Fresnel equations for a constant material in the scene	Fresnel equations for different material layers
Material attenuation	No material penetration	Calculated by ray distances inside voxels
Radiation pattern	Spotlight with main lobe calculation of [TSI*06]	Blurring kernel
Considered radiation paths	Single bounce at first material surface	Penetration of materials in transmittance, a reflected ray is sent to transceiver at each material interface on the incoming ray direction

Table 4.2: The differences between [Kli12] and a THz simulation with SVTs are summarized.

distance, the treatment of interference by accumulating ray contributions and the Fresnel equations.

4.5.2.4 *Results*

Properties of a mechanical scanner serve as input

To evaluate the discussed adaptations to the SVT processing, the pixelwise scanning system of Sec. 5.3 is simulated. Here, a transceiver and a receiver are used to scan the scene sequentially. In the rendered SVT images, each pixel represents a discrete position of this transceiver-receiver-combination. The parameters of the system serve as input parameters for the simulation. Tab. 4.3 provides an overview of these parameters.

Description	Symbol	Value
Frequency	f	578.32 GHz
Wavelength	λ	0.52 mm
Lens radius	r_{lens}	20.32 mm
Num. aperture	NA	0.508
Beam waist	ω_0	0.31 mm
Rayleigh length	z_r	0.59 mm

Table 4.3: Setup properties for simulating THz radiation with SVT rendering

Gaussian kernel and 100x100 rays per pixel are used

Further input properties are the radiation pattern and the number of rays per pixel. The radiation pattern is given by a Gaussian kernel as described in Sec. 4.5.2.2. For the determination of a plausible ray sampling per pixel, the approximation of $\lambda/10$ (see [LCL89]) leads to a recommended sampling of 12x12 rays per pixel in the focus (see Eq. 4.34) and 782x782 rays per pixel at the lens (see Eq. 4.35). Since the performance heavily relies on the number of rays, simulated phase maps with a varying number of rays have been compared to reduce the computation time and keep the correctness of the results. Hence, a reflection from a plate has been analyzed. Fig. 4.18 shows the result. It can be seen that the reconstruction of the phases improves with an increasing number of rays. Between 90x90 and 100x100, the phase maps do not change noteworthy, so that 100x100 samples are used for further evaluation.

$$2 \cdot \omega_0/(\lambda/10) = \lceil 11.92 \rceil \tag{4.34}$$

$$2 \cdot r_{lens}/(\lambda/10) = \lceil 781.54 \rceil \tag{4.35}$$

Information on the four test scenes

The four different test scenes *Material*, *Rough*, *Usaf* and *Crystal* are used as input to show different influences on the THz simulation. Fig. 4.19 provides renderings and information on spatial properties for these scenes. Those SVTs are created from surface meshes by the technique of Sec. 4.5.2.1. All scenes have a resolution of 4000 voxels in each dimension. This leads to a scene resolution of 50 µm for the scenes *Material*, *Rough* and *Usaf*. The *Crystal* scene requires a smaller scale, so that the resolution becomes 5 µm. This scene resolution is

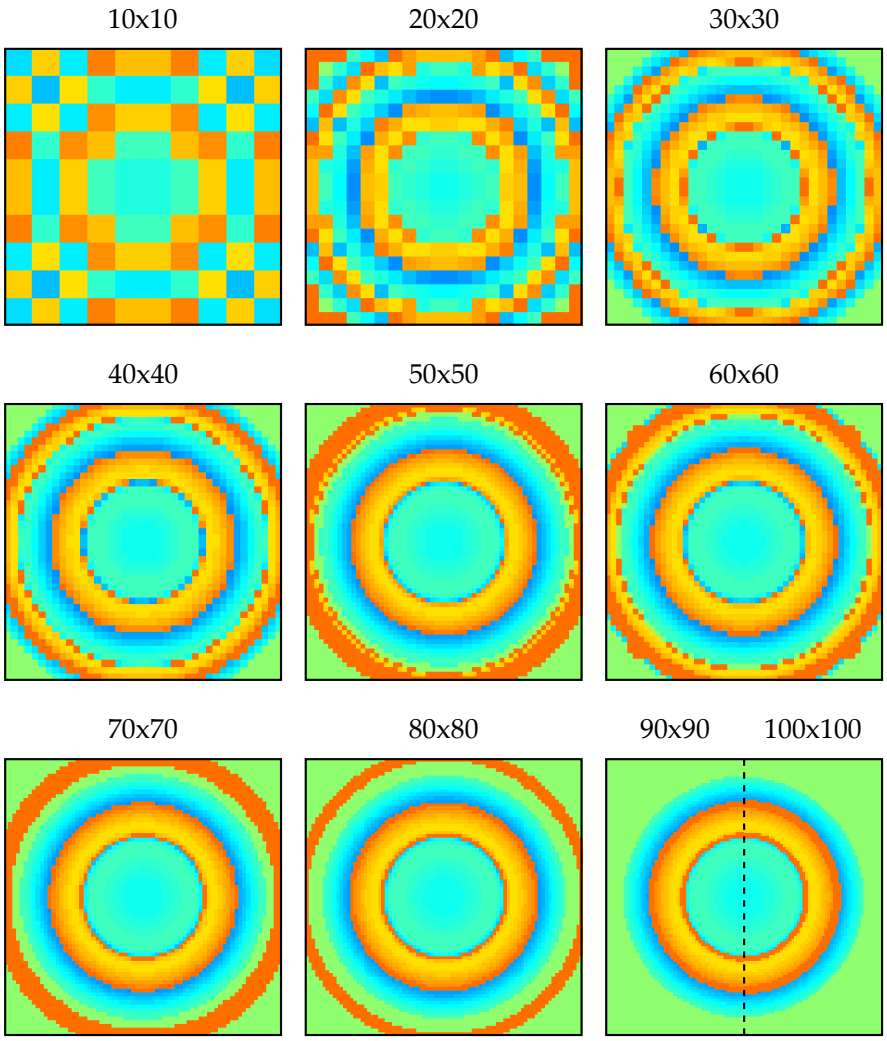


Figure 4.18: A series of color-coded phase maps is shown. The focusing simulation of Sec. 4.5.2.2 is used (cf. Fig. 4.16). Reflected phases from an obstacle are calculated per ray. The rays per pixel increase from 10x10 in the upper left corner to 100x100 in the bottom right corner. It can be seen, that the ray sampling consolidates between 90x90 and 100x100 pixels, so that following measurements are done by 100x100 rays per pixel.

the quantization error of the SVT creation (cf. Sec. 4.5.2.1). Statistics of the SVT creation and the performance of the simulation are given in Tab. 4.4. Since all SVTs are processed at a resolution of 1280x720 pixels, the lateral spatial resolution varies as well. For *Material*, *Rough* and *Usaf*, a distance of 240 µm is used. The *Crystal* scene is sampled with transceiver distances of 12 µm.

Material properties and thickness are evaluated in Material scene

The *Material* scene contains 9 plates with an extent of 4x4 cm per plate. The thicknesses of these plates vary between 1 mm, 2 mm and 3 mm. Furthermore, the plates consist of varying materials. The used

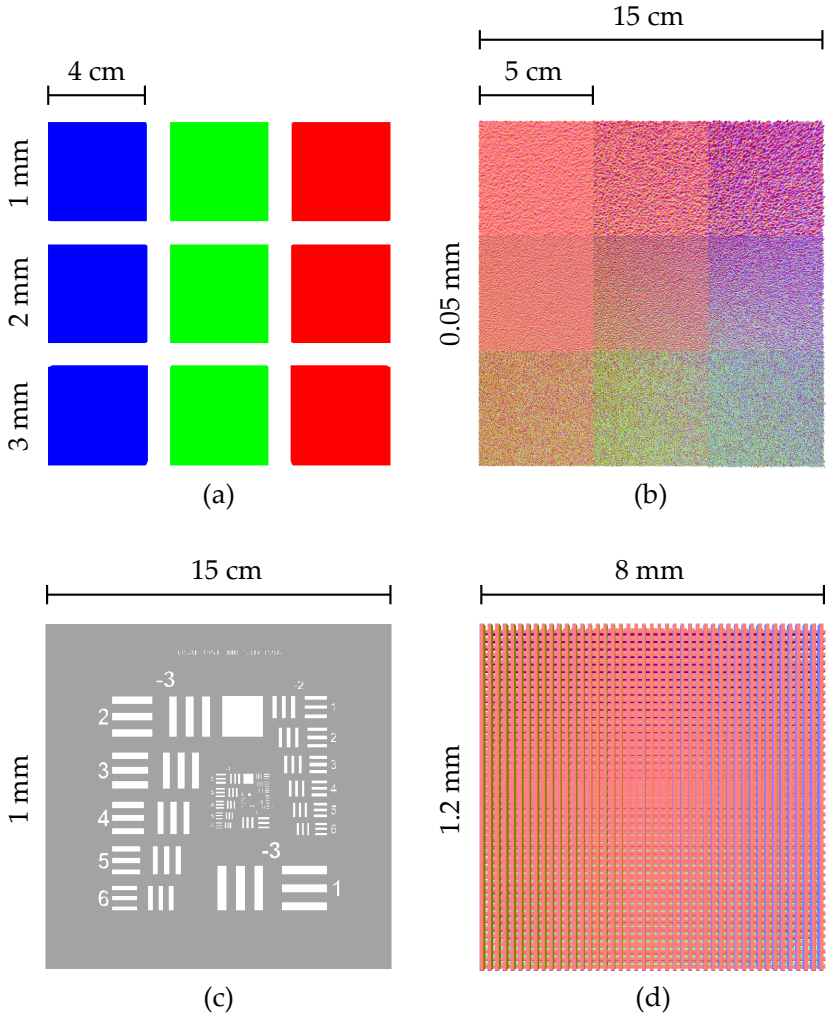


Figure 4.19: SVT renderings of the used test scenes are shown. While (a) and (c) show color values, (b) and (d) show a colored normal representation. The upper sizes show the scale of the scene. The left sizes provide thicknesses of the scene. The *Material* scene contains 9 plates with varying materials and thicknesses. From left to right, pure cellulose paper, polystyrene and polycarbonate are used. From top to bottom, the thicknesses of the plates are 1 mm, 2 mm and 3 mm (a). The *Rough* scene contains a plate with 9 regions which hold different roughness properties. From left to right, the correlation length L_c is set to 1 mm, 2.5 mm and 5 mm. From top to bottom, the standard deviation of surface heights σ_h is set to 1 mm, 2.5 mm and 5 mm. The plate is treated as a perfect reflector (b). The *Usaf* scene contains a variant of the USAF resolution test chart, which is commonly used for the evaluation of optical imaging systems. It is implemented as a perfect reflector (c). The *Crystal* scene simulates the photonic crystal structure of [GMM*02]. The material of the scene is silicon. The single bars of the grid have a thickness of 78 µm and are approximated by thicknesses of 80 µm or 85 µm due to the voxel resolution of 5 µm in this scene (d).

Scene	#Tri. [M]	#Vox. [M]	Size [MB]	SVT [s]	Sim. [m]
Material	18.4	236.9	952.7	98.2	13.5
Rough	112.5	38.7	177.1	112.5	12.5
Usaf	35.7	165.3	667.8	182.8	11.8
Crystal	64.8	321.3	1307.3	129.0	75.0

Table 4.4: From left to right column, scene name, number of processed triangles, number of generated leaf voxels, total memory consumption, performance for creating the SVT and simulation performance (10000 rays per pixel at a resolution of 1280x720 pixels) of the four test scenes are provided. While the other timings are very similar, the performance of the simulation for the *Crystal* scene drops significantly due to the complex geometrical structure. Here, the grid layers lead to the creation of much more rays during the traversal.

materials and the respective indices of refraction at $f = 578.32$ GHz are:

- pure celullose paper: $1.4 - i0.047$
- polystyrene: $2.11 - i0.002$
- polycarbonate: $1.64 - i0.012$

Tab. 4.5 provides the results for transmittance and reflection. Furthermore, the simulations with one ray per pixel and with 100x100 rays per pixel are compared. It can be seen that the transmitted signals become weaker with an increasing material thickness. Additionally, the difference between one ray and 100x100 rays per pixel is negligible for transmittance. For reflection, the values differ more strongly and a linear dependence on the thickness is not recognizable because two reflected signals interfere and further reflection paths are omitted in the simulation. The difference in the reflection between one ray and 100x100 rays per pixel is stronger and leads to more varying values. Here, the accumulation of primary and reflected rays changes the signal strength due to the superposition of ray radiations with different path lengths.

Rough scene shows the influence of explicit roughness representations

The influence of an explicit roughness representation by voxels is evaluated in the *Rough* scene. A perfectly reflecting plate with nine regions of different roughnesses has been created. Similar to [Kli12] (cf. Sec. 4.5.1.2), the 2D-generation of surface heights of [BPK07, BPK08] has been used to create height values. Afterwards, the resulting surface points were triangulated. After a voxelization of this mesh, the SVT has been used in the THz simulation. Fig. 4.20 shows the rendered simulation result. All combinations for $L_c = 1.0$ mm, 2.5 mm, 5.0 mm and $\sigma_h = 1.0$ mm, 2.5 mm, 5.0 mm have been examined. To obtain a more informative interpretation of the result, the values of a

	Reflection			Transmittance		
	PCP	PS	PC	PCP	PS	PC
	1 ray per pixel					
1 mm	−14.14	−3.78	−8.63	−5.49	−1.41	−1.89
2 mm	−16.2	−5.53	−18.21	−10.48	−1.63	−3.2
3 mm	−15.22	−9.9	−10.84	−15.48	−1.85	−4.5
	10000 rays per pixel					
1 mm	−15.44	−6.43	−11.35	−5.64	−1.69	−2.03
2 mm	−15.39	−10.61	−13	−11.02	−3.49	−3.92
3 mm	−15.37	−10.1	−11.48	−16.57	−6.19	−6.26

Table 4.5: The resulting signals of the *Material* scene are shown. Pure cellulose paper (PCP), polystyrene (PS) and polycarbonate (PC) are simulated with thicknesses of 1 mm, 2 mm and 3 mm. Values for transmittance and reflection are given for a simulation with 1 and 10000 rays per pixel. The values are provided in dB.

window of 100x100 pixels of each region have been averaged. Tab. 4.6 shows the result of these values. It can be seen that the smoothest roughness region with $L_c = 1.0$ mm and $\sigma_h = 1.0$ mm reflects with -6 dB the most radiation in the direction of the transceiver. While the reflected signal drops to -15.87 dB for $L_c = 1$ mm and an increasing σ_h, it remains constant at -23 dB for $L_c = 5$ mm, although σ_h increases in the same manner. Here, it can be seen, that the ratio between both values influences the signal as well. Furthermore, it is shown that the roughest surfaces lead to the weakest signals.

Focusing and radiation patterns are analyzed in the Usaf scene

The behavior of beam focusing is evaluated in the *Usaf* scene, which contains a USAF resolution test chart. The material of the chart acts as a perfect reflector. Fig. 4.21 (a) shows the simulated result for a chart which is slightly tilted from the beam direction. While the diagonal axis with 0 dB holds the surface normals which coincide with the beam direction, the signal decreases to roughly -15 dB in the

		L_c in mm		
		1	2.5	5
σ_h in mm	1	−5.99	−16.40	−23.04
	2.5	−10.43	−19.52	−23.25
	5	−15.87	−21.08	−23.61

Table 4.6: The signals of a window with 100x100 pixels have been averaged for each roughness combination of the *Rough* scene. The resulting values are provided in dB.

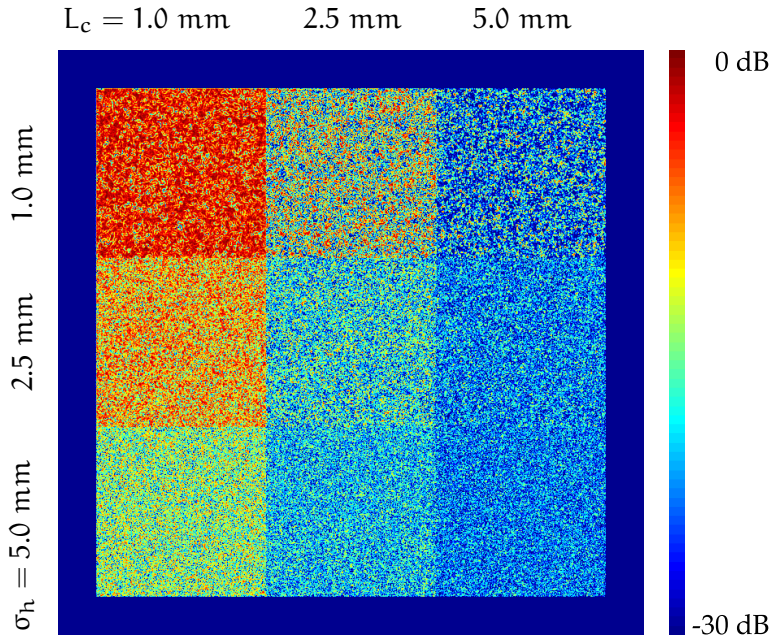


Figure 4.20: A plate with varying roughnesses is shown in the *Rough* scene. While the smoothest part in the top left corner reflects most strongly, the reflected signal of the roughest part in the bottom right corner is the weakest.

top left and the bottom right corner due to surface normals which deviate from the beam direction. Furthermore, a blurring of the signal can be perceived at these corners, because the rays of a beam intersect the plate at different depths. Fig. 4.21 (b) and (c) compare the simulation with two different radiation patterns. While a Gaussian kernel is used in Fig. 4.21 (b), a box filter is used in Fig. 4.21 (c). It is observable that Fig. 4.21 (b) is sharper, because the inner rays of the beam have a larger contribution to the final signal than the outer rays. In Fig. 4.21 (c), signals of outer rays have a higher influence, because all rays contribute equally to the final result.

Modification of frequency and lens radius is evaluated in the Crystal scene

The *Crystal* scene is used to evaluate the frequency-dependency and the influence of lens radius r_{lens}. It contains the photonic crystal of [GMM*02]. The grid structure consists of silicon which is simulated by an index of refraction of $3.42 - i0$. Fig. 4.22 and 4.23 show the signal behavior for a varying frequency between 350 and 850 GHz in reflection and transmittance, respectively. While the reflected signal of Fig. 4.22 drops to approximately -30 dB between 450 GHz and 650 GHz, it remains around -10 dB for the other frequencies. Fig. 4.23 shows the transmitted signal, which decreases for an increasing frequency. The measurement of [GMM*02] shows a frequency gap between 400 and 550 GHz for transmittance. It follows that the SVT simulation allows frequency-dependent processing, but it does not coincide with the expected behavior exactly.

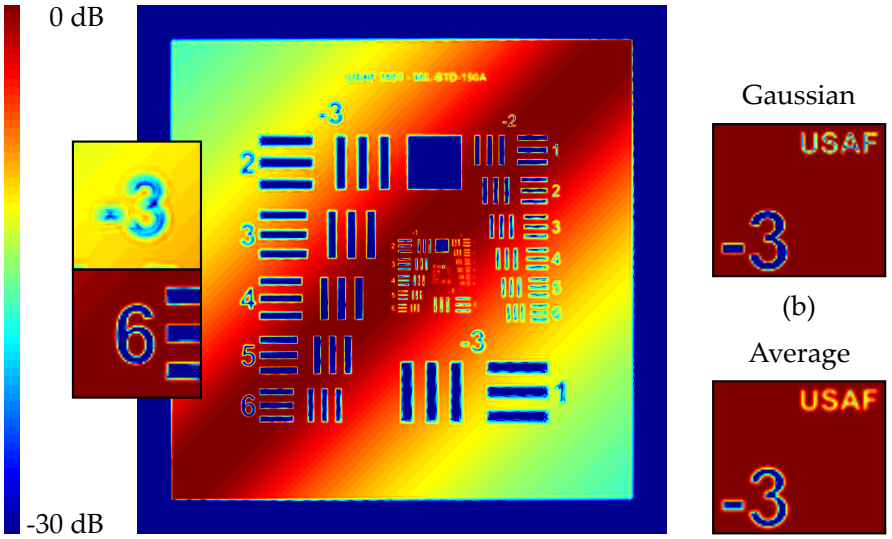


Figure 4.21: The slightly tilted USAF plate of the *Usaf* scene shows the influences of the focusing. The top left and the bottom right corner becomes blurry and the signal weakens. The top right and bottom left corners lie on the axis, which is perpendicular to the scanner direction. The signal is focused and maximized on this axis. To illustrate the blurring, two image regions on the left show a digit in the blurred part and a digit on the sharp axis (a). The kernel for simulating the radiation pattern influences the imaging as well. Two magnifications of the perpendicular USAF plate show the difference between a Gaussian kernel (b) and an average kernel (c). It can be seen that the Gaussian leads to a more focused image.

Reasons for differing result

One reason for this difference is the quantization of the voxelization. Here, the thickness of the real grid bars may not be reached and the thicknesses of different bars vary slightly as well. This behavior leads to a blurring of the signal. Other reasons are the missing ray bending and the incompleteness of the simulated radiation paths (cf. Fig. 4.15 and 4.17). Although, the effect of multiple bounces is observable for the reflection, the ray directions are not changed and the traversal stops so that possible ray contributions are not considered completely. This influence of missing bounces is even more obvious for transmittance, because no additional rays lead to the fact that the measured gap is missing completely. Furthermore, the ratio between geometrical features and used wavelengths leads to a large influence of wave effects in the measurement. Since these effects can not be simulated with a purely ray-based approach, the simulation differs from the measurement as well.

Increasing lens radius reduces the steepness of the frequency gap

To show the influence of r_{lens}, simulations for $r_{lens} = 17.5$ mm, 20 mm, 22.5 mm and 25 mm are compared. For each simulation, the same four pixels are averaged. The result is depicted in Fig. 4.24.

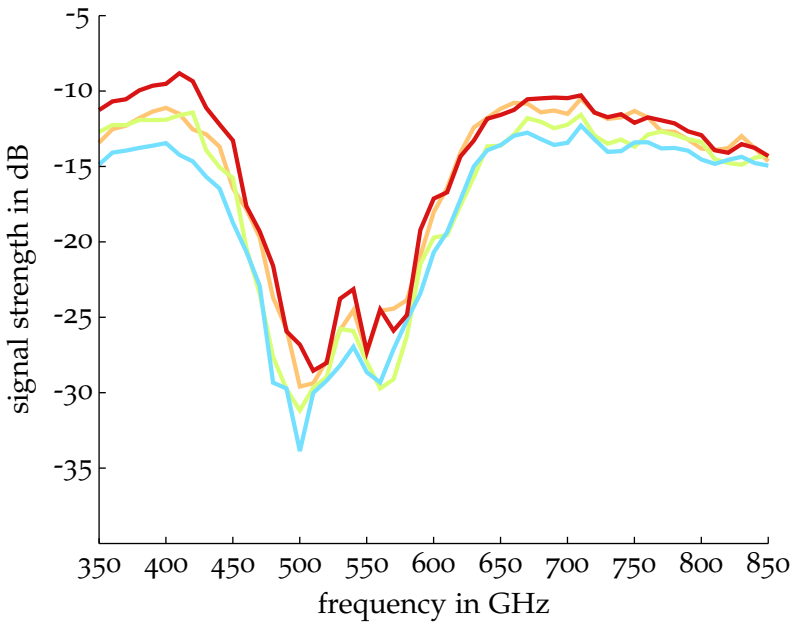


Figure 4.22: The reflected signals of the *Crystal* scene are shown. Four different pixels of the simulated grid are taken. Although the results differ from the measured values of [GMM*02], a frequency gap is created due to geometrical features and the interference of multiple reflection rays.

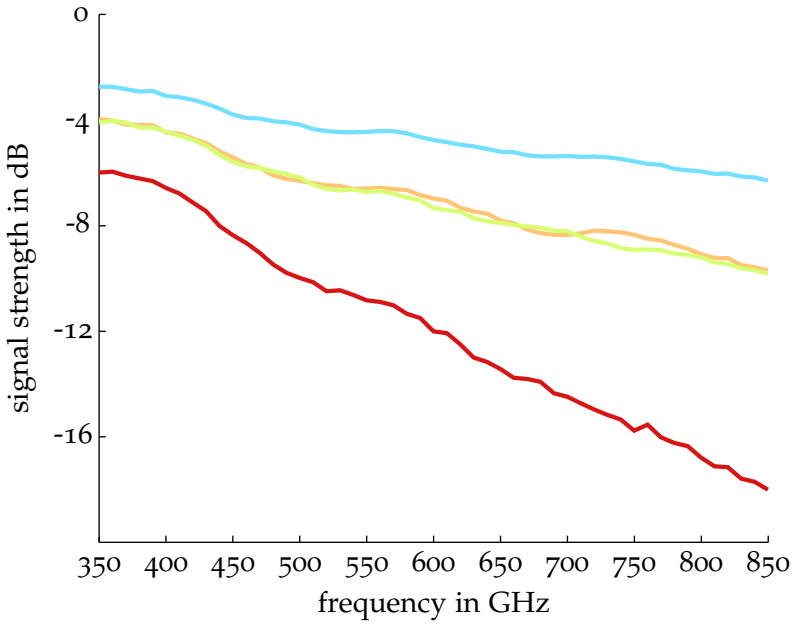


Figure 4.23: The transmitted signals of the *Crystal* scene are shown. Four different pixels of the simulated grid are taken (Colors represent the same pixels as in Fig. 4.22). The results differ from the measured values of [GMM*02] and the frequency gap is not reproducible.

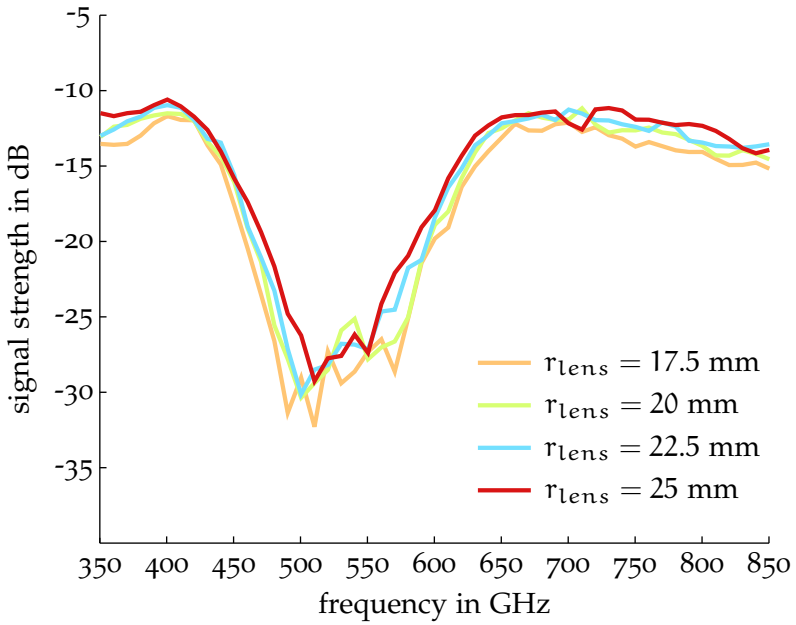


Figure 4.24: The reflection behavior of different lens radii r_{lens} in the *Crystal* scene is depicted. It can be seen that the steepness of the frequency gap is reduced with an increasing r_{lens}. Each chart line holds an average of the same 4 pixel positions. The green line with $r_{lens} = 20$ mm corresponds to the averaged signal of all lines in Fig. 4.22.

With an increasing r_{lens}, the window of the lowest signals in the frequency gap gets smaller and the steepness between the changes from -10 dB to -30 dB is reduced. It is assumed that the increasing variation of incoming ray orientations and ray lengths for a larger r_{lens} reduce the effect of the grid and smooth the immediate change between maximum and minimum of the measured frequency gap.

4.6 SUMMARY

CG methods need to be adapted for THz simulation

In this chapter, it has been covered how the simulation of THz imaging can be realized by CG methods. Common CG techniques and simplifications for performant and accurate simulation of visible light have a restricted applicability in the THz range, so that these methods need to be adapted. After a discussion on the main aspects that need to be considered for a THz simulation, two simulations are introduced and discussed. While the basic mechanisms are similar in both approaches, the underlying THz setups vary strongly, so that different THz properties are simulated.

Properties of [Kli12]

The first simulation of [Kli12] focuses on the imaging simulation for a hybrid THz scanner, which combines a synthetic aperture reconstruction and a mechanical scanning to obtain a 3D representation of the scanned object. To simulate this setup behavior, a common

volume rendering is employed. The traveled ray distances are considered for a calculation of coherent radiation and a local Cook-Torrance model is employed for calculating the reflection mechanisms at the surface of the screened object. Here, the polarization is considered by using Frenet frames and Jones vectors. Furthermore, the influence of surface scattering is incorporated by a statistical model for describing roughness and a spotlight model is used to simulate a simplified radiation pattern of an antenna.

Properties of SVT simulation

In the second simulation, the proposed SVT of Chap. 3 is applied to the simulation of a THz system which operates in transmittance and reflection mode. The implicit properties of the SVT for representing inner structures and fine surface details allow a multi-material simulation and the consideration of attenuation inside materials. Since the scanner uses focused radiation, a lens system is considered in the SVT rendering as well. Here, the tracing of individual rays is performed to simulate the contribution of a focused beam. Additionally, a blurring kernel is used for representing the radiation pattern of the antenna and a first approach for computing multi-bounce effects is proposed.

Physical plausibility is restricted, but performance is increased

Combining both approaches to a more generalized simulation is not trivial, because the complexity increases if the simulated effects influence each other in one model. Therefore, comparisons with true measurements are restricted as well. While the correctness of the simulations is also limited by the ray approximation of geometrical optics, the performance for calculating scenes with negligible influences of wave optics is greatly improved. This performance gain allows a fast and efficient simulation of imaging capabilities of respective THz prototypes.

5 DESCRIPTION OF THZ IMAGING SYSTEMS

While Chap. 3 and 4 focus on general aspects of simulating THz radiation, the subject of this chapter refers to more specific CG solutions for individual THz scanner setups. A motivation for applying those solutions to THz setups is given in Sec. 5.1. It follows a case study for two prototypical scanners. First, the used CG methods for a hybrid 3D scanning system are presented in Sec. 5.2. After a general description of the system, the used CG techniques for a performant multimodal 3D THZ imaging are discussed. This section is based on [PKK*13]. Afterwards, Sec. 5.3 provides the CG principles for a pixelwise scanning system.

5.1 MOTIVATION

THz prototypes require individual solutions

THz imaging systems are still under heavy development and many different scanning prototypes have been built in the last years. Each system requires an individual adaption of used techniques, because different scanning and acquisition procedures need to be exploited for the imaging. Therefore, a general solution which incorporates all possible scanning properties is not reasonable.

CG methods allow an efficient processing

Usually, the built systems acquire a massive amount of data which need to be processed for imaging. Due to the independence of measured points, rows or slices for individual elements in the final scene representation, it is very promising to apply CG methods for data processing because the parallelism of GPUs can be exploited. In comparison to sequential processing of the measured data, a drastically improved performance is possible. Furthermore, usual CG methods are required for an efficient and performant visualization of the reconstructed scene representation.

CG visualization requires a geometrical description

An individual geometrical configuration and calibration for the respective scanning system are required for parallel processing and rendering by CG methods, because the expense of THz scanning hardware leads to varying setups for the scanner operation. For active imaging systems, the geometrical positioning of transmitters and receivers needs to be described by a meaningful representation without redundancies.

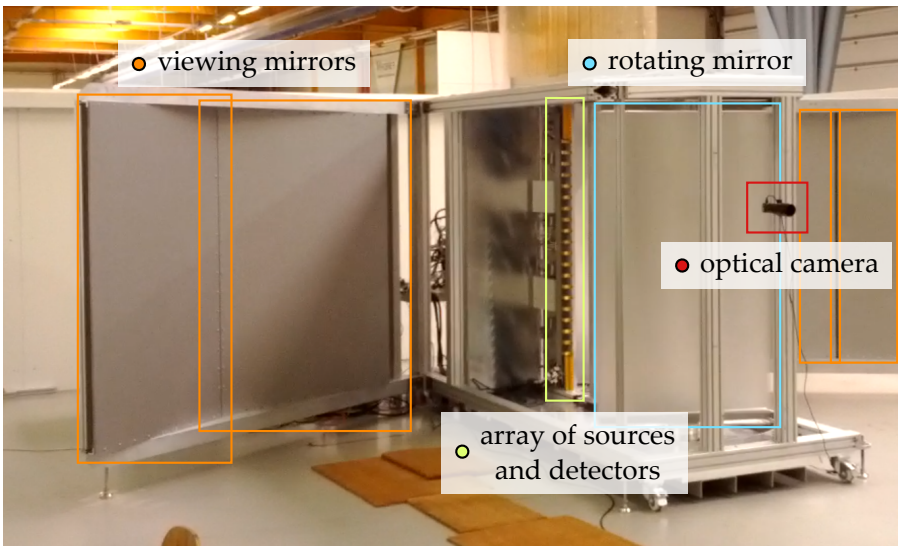


Figure 5.1: The prototypical scanning system for hybrid 3D THZ imaging is shown. The linear array of sources and detectors can be seen in the middle of the image. The rotating mirror and the optical camera are placed on the right side of the image. The left mirrors are used to obtain different viewing angles of the scene.

5.2 HYBRID 3D SCANNING SYSTEM

5.2.1 *System*

*Used scanner is presented in [KKP*12]*

With a focus on the signal processing, the used prototypical scanner for a hybrid 3D THz imaging is described in [KKP*12]. It delivers the input data for the processing steps, which are discussed in the following sections. While Fig. 5.2 shows a conceptual setup which is used for explaining a generalized geometrical description of such a system in Sec. 5.2.2, Fig. 5.1 shows the built scanner.

Imaging parts and functionalities of the THz scanner

A linear array with 20 sources and 24 detectors is built up as a sparse array. The used frequency range lies between 80 and 110 GHz. The array is vertically mounted in order to allow a 2D reconstruction by synthetic aperture imaging along the depth and height direction (see Sec. 5.2.4). First, the horizontal beam patterns of the antenna elements are spread by a small convex cylindrical mirror. Afterwards, they are focused by a wider concave cylindrical mirror. A planar rotating mirror is used to steer the resulting optical focal slice across the screened object (cf. Fig. 5.2). Therefore, the imaging in the lateral direction is done by a mechanical rotation. Additionally, the rotating mirror reflects the radiation to different static mirrors. These mirrors allow different viewing angles of the object to enhance the visibility of the object in the imaging. In comparison to a full 3D reconstruction, such a hybrid approach reduces the computational load significantly by removing one dimension in the synthetic reconstruction. Further-

more, the hardware costs are reduced and the flexibility of the scanner is increased, because no additional THz components are required for the reconstruction in the mechanical dimension. A fusion of the sequentially reconstructed 2D slices leads to the creation of the final 3D scene representation (see Sec. 5.2.5.2).

An additional optical camera enables multimodal imaging

In addition to the THz acquisition of the scene, an optical camera creates a visual 2D representation of the scene as well. The goal of this multimodal acquisition is an overlay of the THz data by the visual representation (see Sec. 5.2.5.3) to protect the privacy of scanned persons if the system is used as a body scanner. To match the created representations, the calibration for obtaining the fused 3D representation is extended by a method for transforming reference points between both modalities (see Sec. 5.2.3).

5.2.2 *Configuration*

Description of workflow

Heterogeneous measurement data need to be processed to allow a full 3D image generation for this hybrid scanning system. Each position or orientation of the mechanically rotating mirror gives one discrete dataset, which needs to be reconstructed independently. This reconstruction is based on backprojection, a standard technique in synthetic aperture imaging. Afterwards, the reconstructed datasets are fused to one consistent 3D representation of the scanned object by using geometrical information of the system. A raycaster visualizes the resulting volume grid.

general	type of setup (angular or parallel slices)
	#views, #slices, #transmitters and #receivers
	angle offset between consecutive slices (angular)
	positions of the transmitters and receivers
	frequency range and number of samples
each view	position of the center of projection
	direction of the first slice
	distance offset between center of projection and unfolded position of tr-array
bounding box	position of center of bounding box
	width, height and depth of the bounding box
	resolution of the bounding box

Table 5.1: Parameters for the system configuration are shown. They can be categorized into general and view-dependent parameters. Furthermore, the bounding box of the screened volume is parameterized. Source: [PKK*13]

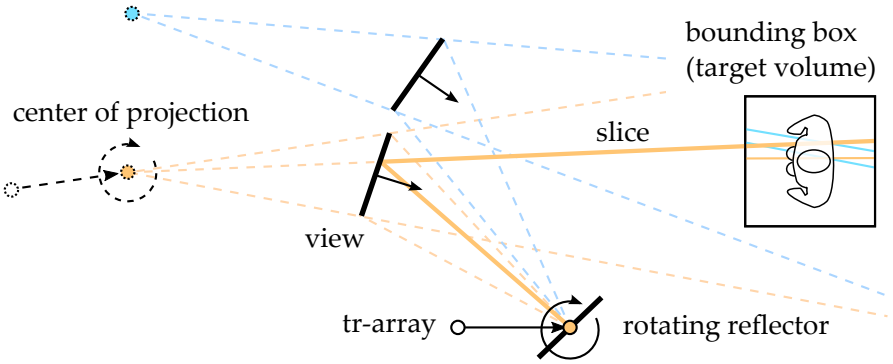


Figure 5.2: The concept for the spatial definition of an exemplary THz setup is shown. The position of the rotating reflector is unfolded at the static reflectors. It defines a view with a center of projection for each static reflector. An infinitesimal slice approximates the beam of the antennas. The position of the tr-array is virtually placed by the opposite direction of each slice starting at the center of projection and a fixed distance between tr-array and rotating reflector. The slices inside the bounding box hold the information of the reconstructed shape of the target. Image source: [PKK*13]

Configuration for 2D reconstruction and 3D fusion

To describe the system properties unambiguously, a single configuration file is used for all steps of the processing workflow. Tab. 5.1 shows the parameters of this configuration file. The general parameters are mainly used for reconstructing the individual synthetic aperture images with spatial intensities in two dimensions. The description of the geometrical system properties is an essential part for a geometrically correct positioning and the fusion of the reconstructed data. Therefore, a concept for efficiently describing the spatial system properties is introduced. Fig. 5.2 illustrates the used terminology of the concept, which is described in the following.

Definition of tr-array, slice and bounding box

A scanner contains transmitters and receivers to create signal data. These antenna elements are positioned on a linear array (tr-array). The path of the THz radiation which is emitted and received at the tr-array is approximated as a discrete plane (slice) by the assumption of a perfectly focused beam in the dimension of the mechanical scanning. The radiation is sent to a volume (bounding box) containing the observable object and reflected back along the slice direction to the receivers of the tr-array. The bounding box holds the final volume representation of the fused 3D representation and defines the size of the slices, because reconstructed values are only considered if they are inside the bounding box.

Concept of unfolding and views

Mechanical scanning as a column by column, or line by line image generation, leads to the interpretation that slices which hit the same static reflector along the whole radiation path define a view of the target. If the mechanical scanning is based on a rotating element, it is

possible to unfold the slices at each reflector and determine a center of projection, which allows to calculate virtual positions of the tr-array for every slice. For a mechanical scanning that creates parallel slices, only a linear offset between the virtual positions of the tr-array is needed.

Used parameters allow an efficient calculation

These virtual positions of the tr-array are calculated because the global position of every slice needs to be determined in a common coordinate system to achieve the spatial matching to the 3D object and an accurate fusion of the different viewing directions. A slice is represented by a direction, the bounding box position and the unfolded position of the tr-array. While the direction and the unfolded position of the tr-array vary for each slice, the bounding box is the same for all slices. Depending on the mechanical movement of the THz components, the direction and the unfolded tr-array position change differently. In setups with a movement that creates parallel slices, the direction remains the same, but the unfolded tr-array position is translated. With a mechanical rotation of a reflector, a center of projection remains static while the direction and the unfolded tr-array position change for every slice. To describe all slices of a view for spatial positioning in the case of a rotating reflector, it is sufficient to store a center of projection with a linear offset to the tr-array, a direction for the first slice and an angular offset between slices.

5.2.3 *Geometrical Calibration*

5.2.3.1 *Overview*

Configuration is obtained from calibration

While the geometrical configuration is a conceptual description of the scanner, the geometrical calibration determines the correct parameters for this description. Since the bounding box is a virtual concept and no part of the scanner, the properties for the geometrical configuration can not be obtained from the scanner specification.

Geometrical calibration depends on signal calibration

In the startup procedure of the scanner, a manual calibration of the signal processing is executed. After this step, it is possible to reconstruct the correct depth values between tr-array and reflecting object. The object needs to be in a defined bounding box to avoid ambiguities (cf. [KLD*10]) which originate from the reconstruction method (see Sec. 5.2.4). This calibration procedure of the signal processing serves as a basic precondition of the proposed geometrical calibration which is described in the following. If this calibration is not done correctly, the succeeding geometrical calibration can get erroneous.

Steps of the geometrical calibration

Under the assumption of correct depth values, reference points inside the bounding box can be determined. They need to be described in scanner coordinates and reconstructed by the scanner to obtain the correct configuration parameters. To allow this transformation, a measurement of reference points by a laser distance meter (see Sec. 5.2.3.2) is performed. It allows to transform points inside the bounding box

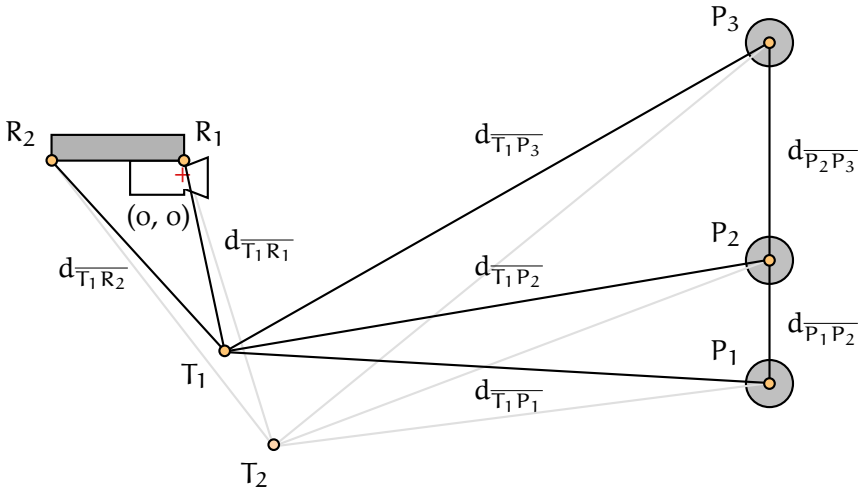


Figure 5.3: The setup for the measurement and the corresponding parameters are shown. For clarity, the parameters for T_2 and $d_{\overline{P_1P_3}}$ ($= d_{\overline{P_1P_2}} + d_{\overline{P_2P_3}}$) are omitted.

to scanner coordinates. After a scanning of the same scene, configuration parameters are derived from the information of the reconstruction (see Sec. 5.2.3.3). In an additional step, the obtained coordinates need to be transformed to the coordinate system of the optical camera for a multimodal fusion of optical and THz data (see Sec. 5.2.3.4).

5.2.3.2 *Measurements with a Laser Distance Meter*

Scanner coordinates are calculated by optimization

To obtain points in the bounding box which are transformed to scanner coordinates, three points in the bounding box P_i with $i = 1, 2, 3$, two reference points R_k with $k = 1, 2$ at the scanner and two positions of a laser distance meter T_j with $j = 1, 2$ are used to formulate an optimization problem. The distances d between the positions of the laser distance meter and the other points lead to Eq. 5.1, Eq. 5.2 and Eq. 5.3. Solving this system of equations by optimization leads to the scanner coordinates of P_i. Fig. 5.3 gives an overview of the measurement setup.

$$\|T_j\|^2 - 2(T_j \cdot P_i) + \|P_i\|^2 - d^2_{\overline{T_j P_i}} = 0 \tag{5.1}$$

$$\|T_j\|^2 - 2(T_j \cdot R_k) + \|R_k\|^2 - d^2_{\overline{T_j R_k}} = 0 \tag{5.2}$$

$$\|P_h - P_i\| - d_{\overline{P_h P_i}} = 0 \text{ with } h \neq i \tag{5.3}$$

Requirements for reference pillars

The points in the bounding box are represented by 3 vertically oriented pillars. Fig. 5.4 shows a photo of these pillars. Since they are needed for the determination of further scanner settings as well, the following additional requirements need to be fulfilled. For a unique and less error-prone identification in the reconstruction, they need to be placed on a straight line with varying distances between each

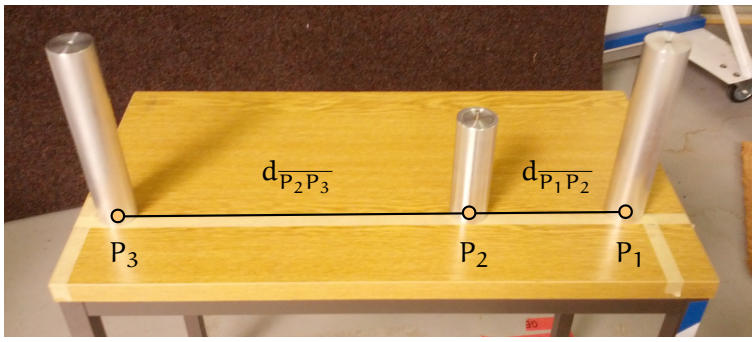


Figure 5.4: A photo of the scanned pillar scene is shown. The metallic pillars are on a straight line and have varying distances between each other. The pillar parameters can be seen again (cf. Fig. 5.3).

other. Furthermore, they need to be placed inside the desired bounding box so that all views can detect all pillars. To guarantee a reliable reconstruction, the pillars should consist of a material which is highly reflective in the THz range. In the realized calibrations, metallic pillars are used. Another important factor is the orientation of the pillars to the scanner. For a more precise matching between measurements of laser distance meter and reconstructed values, the pillars need to be parallel to tr-array, rotational mirror and viewing mirror, because the distance between reflected object and tr-array must be unambiguous.

5.2.3.3 *Determination of Configuration Parameters*

Required configuration parameters

View parameters (v) and the position of the bounding box need to the obtained from the geometrical calibration. All other configuration parameters of Tab. 5.1 are independent of the scanner positioning and can not be calculated in this step. Therefore, the following properties are determined:

- position of the center of the bounding box P_{bbox}
- unfolded center of projection U_v
- direction of first slice $\vec{f_v}$
- distance offset $\Delta d_{v,off}$ between U_v and unfolded position of tr-array A_v

Obtained parameters from pillar reconstruction

The value of P_{bbox} is set to P_2, because it is the middle pillar of the already calculated reference points and other points would need an additional transformation to scanner coordinates. The other parameters require the reconstruction of the same pillar scene which is used for the measurement by the laser distance meter already (see Fig. 5.4 and Fig. 5.3). After the pillar scene has been acquired and reconstructed for all views, the individual slices $s_{v,m}$ are analyzed to

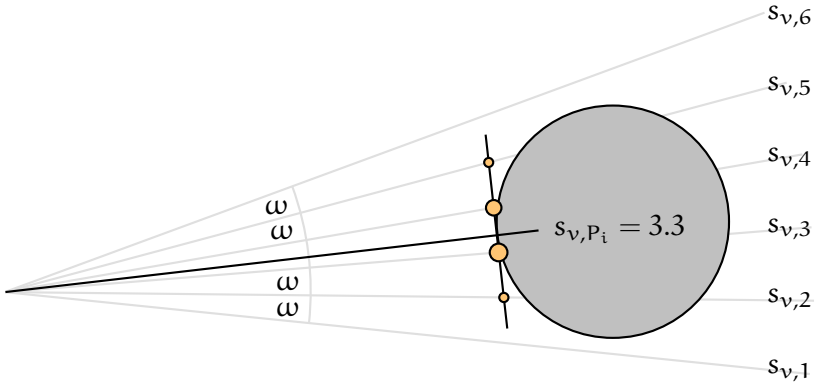


Figure 5.5: An example for discrete slices with varying intensities from the pillar is shown (size of orange dots). The theoretical slice which has the maximum intensity is determined by a curve-fitting from the slices $s_{\nu,m}$. It allows to calculate angular offsets $\varphi_{V_n,P_h\to P_i}$ between the pillars.

receive distances d_{ν,P_i} and angles $\varphi_{\nu,P_h\to P_i}$ (see Fig. 5.6). While d_{ν,P_i} represents the distances on the slices between start of the slice and pillar positions, $\varphi_{\nu,P_h\to P_i}$ represents the angles between the pillar positions.

Curve-fitting gives theoretical angle of pillar centers

The reconstructed intensities of one pillar are found in several slices and vary depending on the reflection direction of the pillar. To find the highest intensity which represents the center with the perpendicular reflection direction of the pillar, the intensities of the discrete slices ($s_{\nu,m}$ with $m \in \mathbb{N}^+, 1 \leqslant m \leqslant M$) are estimated by curve-fitting to get the theoretical slice $s_{\nu,P_i} \in \mathbb{R}_{>1}^{\leq M}$ with the highest reflection. (see Fig. 5.5).

Determination of angles between pillars

With the constant angle offset ω between consecutive slices, which is obtained from the motor movement of the rotating mirror, it is possible to calculate $\varphi_{V_n,P_h\to P_i}$ as $|s_{\nu,P_h} - s_{\nu,P_i}| \cdot \omega$. Additionally, the nearest slice to the maximum intensity gives d_{ν,P_i}. Further interpolations are not necessary for the distance in the practical test, because the resolution of the reconstruction grid is lower than the curvature of the pillars, so that d_{ν,P_i} is constant in the slices. If the resolution would increase, an additional curve-fitting should be applied.

Calculation of U_ν

Like the determination of P_1, P_2 and P_3, U_ν is calculated by a formulation of an optimization problem. A distance requirement (see Eq. 5.4) and an angle requirement (see Eq. 5.5) are defined. Fig. 5.6 gives an overview of the parameters. Δd_ν is an unknown distance value which describes the distance between U_ν and the start of a slice. This value is constant for all slices of one view. It is obtained from the optimization as well.

$$\|U_\nu\|^2 - 2(U_\nu \cdot P_i) + \|P_i\|^2 - (\Delta d_\nu + d_{\nu,P_i})^2 = 0 \tag{5.4}$$

$$\sphericalangle(U_\nu\vec{P}_h, U_\nu\vec{P}_i) - \varphi_{\nu,P_h\to P_i} = 0 \tag{5.5}$$

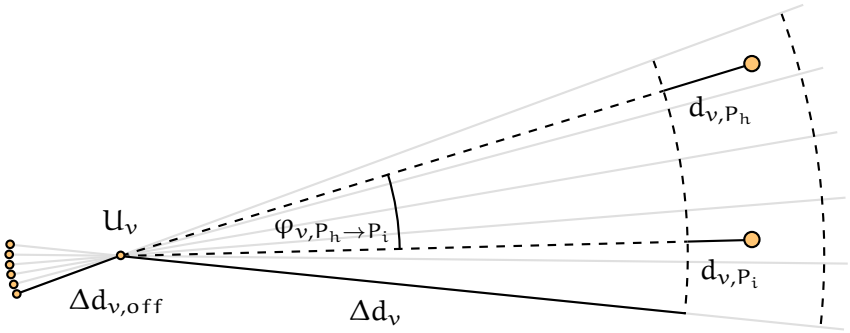


Figure 5.6: The used parameters for determining the unfolded center of projection U_v are depicted. The optimization problem is formulated by Eq. 5.4 and Eq. 5.5. d_{v,P_i} is the distance offset in the slice between start depth and pillar depth in the slice. $\varphi_{v,P_h \to P_i}$ represent the angle between two pillars P_h and P_i. The distance between U_v and the virtual position of the tr-array is given by $\Delta d_{v,off}$. Δd_v is the distance between U_v and the start of the slices. It is unknown and needs to be determined by the optimization as well.

Calculation of $\vec{f_v}$ and $\Delta d_{v,off}$

The direction of the first slice $\vec{f_v}$ is calculated by rotating $U_v\vec{P_i}$ with $-((s_{v,P_i} - 1) \cdot \omega)$. $\Delta d_{v,off}$ should be constant, because the path between tr-array and rotating mirror is the same for all views, but practical tests showed that the signal calibration for the correct depth d_v between tr-array and center of the reconstruction slice does not coincide with the measurement of the laser distance meter followed by the optimization. To compensate this error and obtain the right depth values for a correct fusion of the data, Eq. 5.6 is applied.

$$\Delta d_{v,off} = d_v - \Delta d_v - d_{v,P_{bbox}} \tag{5.6}$$

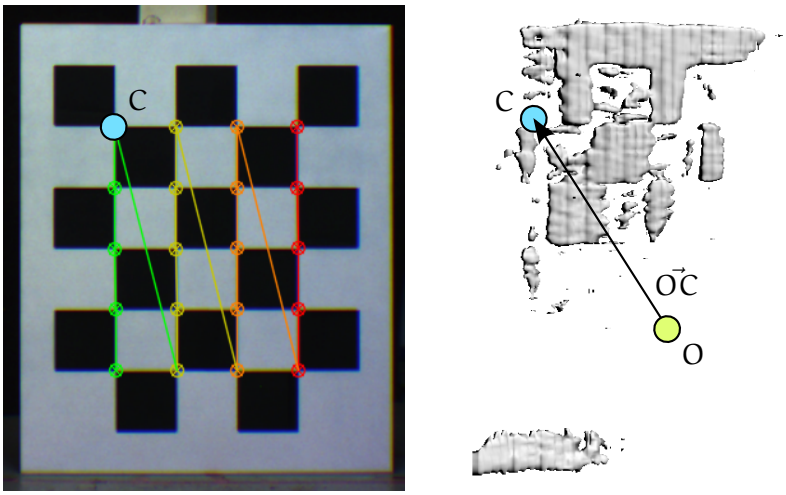
5.2.3.4 *Determination of Optical Coordinate Transformations*

Multimodal fusion requires optical camera coordinates

After the determination of the calibration parameters, every reconstructed point inside the bounding box can be addressed in a global THz coordinate system, i.e. the spatial transformation between tr-array and bounding box is given. For a correct projection of a visual 2D representation onto the THz data in the visualization, an additional transformation between bounding box and optical camera is required.

One calibration object for optical and THz range

To calculate this transformation, a calibration object is created. It needs to be detectable in visual and THz range to allow the detection of point correspondences between both modalities. Therefore, a white metallic board has been created which has holes to form a checker board pattern (see Fig. 5.7 (a)). While a high contrast for THz imaging is achieved by strongly reflecting metal and air which is not reflecting, common computer vision frameworks are optimized for a



(a) (b)

Figure 5.7: A white metallic checker board is used for calibrating the optical camera. A screenshot of the OpenCV calibration is shown (a). The THz representation of the board allows to find point correspondences C between optical and THz coordinate systems. A translation from the origin O is used to transform the THz coordinates to camera coordinates (b).

calibration by black and white checker board patterns in the visual range.

Transformation between coordinate systems required

The transformation between THz coordinates and optical camera coordinates corresponds to the first two transformations in the rendering pipeline between local and camera coordinates (see Sec. 2.1.2.1 and Fig. 2.4). First, the local coordinates of the checker board need to be transformed to global coordinates of the THz scanner. Second, the global coordinates need to be described by coordinates of the optical camera.

OpenCV is used

Both transformations are calculated by using the computer vision framework OpenCV[1] which contains standard algorithms for optical calibration. It is used to calculate the extrinsic and intrinsic parameters of the optical camera, which are represented by matrices. While the extrinsic parameters $\mathtt{Cam}_{\mathtt{ext}}$ give the coordinate transformation between a reference point C (see Fig. 5.7 (a)) on the checker board and the origin of the camera, the intrinsic parameters $\mathtt{Cam}_{\mathtt{int}}$ provide a transformation which relates to the camera properties like focal length or sensor format.

Manual translation between checker board and bounding box

Since OpenCV defines C for calibration, this point is used for the transformation between local and global coordinates as well. Here, the vector $\vec{OC}$ transforms the local origin of the checker board to global scanner coordinates by a translation of the center of the bounding box P_{bbox} which is used as origin O in the visualization. $\vec{OC}$ is

1 http://www.opencv.org

obtained by a visual matching, because C needs to be identified manually in the THz representation of the checker board (see Fig. 5.7 (b)). In summary, a point p in scanner coordinates is transformed by Eq. 5.7 to point p' in coordinates of the optical camera. p' is processed further in the visualization for a correct overlay (see Sec. 5.2.5.3).

$$p' = Cam_{int} \cdot Cam_{ext} \cdot (\vec{p} + \vec{OC}) \tag{5.7}$$

5.2.4 *Reconstruction on GPU*

5.2.4.1 *Overview*

A filtered backprojection is used for the 2D synthetic imaging

Methods of synthetic aperture radar and interferometry are used for imaging of the scanning system. [KLD*10] provides a survey on different methods for terahertz imaging. In the implemented framework, the 2D synthetic imaging dimension is obtained by a filtered backprojection. This backprojection can be improved or replaced easily by other reconstruction techniques which use phase information as well.

Reconstruction properties of the system

In comparison to current reconstruction methods for THz imaging with sparse arrays (see [ASS11, ZB11]), no synthetic reconstruction is used in the lateral direction. Instead, a mechanically rotating mirror reflects the THz radiation and scans the scene in discrete steps, so that a geometrical configuration (cf. Fig. 5.2 and Sec. 5.2.2) and data fusion techniques (cf. Sec. 5.2.5.2) are applied to obtain the final 3D shape from the individually reconstructed 2D slices. The reconstruction has the highest computational effort in the processing workflow because one reconstructed intensity value requires the information of all frequency samples from all transmitter-receiver-combinations. An approach for processing on GPU is proposed to reduce the computation time and to allow a near real-time reconstruction.

5.2.4.2 *Theory*

Theoretical signals are correlated with measured signals to obtain an intensity

The measured frequency samples of each combination of a transmitter and a receiver are needed as input for each 2D slice. The data of each slice are independent from the data of the other slices and can be processed individually. A regular grid represents each 2D slice. Since the contributions for each point on that grid are independent from each other, they are processed individually as well. First, a theoretical phase for each frequency sample $f_{k,tr}$ is calculated from distances d between every point $p(x, y)$ on the slice and each transmitter-receiver-combination tr. Second, phases $\varphi_{k,tr}$ and amplitudes $A_{k,tr}$ are given

by the measurement to correlate the given values in the signal model. Eq. 5.8 shows the corresponding formula.

$$C(x,y) = \sum_{tr=1}^{t\cdot r} \sum_{k=1}^{K} A_{k,tr} \cdot e^{-i\varphi_{k,tr}} \cdot e^{i2\pi f_{k,tr} \frac{d_{tr\to p(x,y)}}{c}} \tag{5.8}$$

Correlation has a high computational effort

The correlation C is the probability that an object is found. Therefore, it can be interpreted as an intensity. A real-time processing is not possible with a straightforward implementation of this calculation, e.g. $32 \cdot 10^{10}$ correlations are needed to reconstruct four views with 100 slices at a resolution of 200x100 points, if the system has 400 transmitter-receiver-combinations and acquires 100 frequency samples. For this example, the implementation requires approximately 130 minutes with a MATLAB implementation on CPU (Intel Core i7 2.8 GHz) and 30 minutes with a CUDA implementation on GPU (Nvidia GTX 480).

Reformulation by a Fourier representation

If the inner sum of Eq. 5.8 is expressed by an inverse Fourier transformation, it can be used to accelerate the backprojection reconstruction method of [Sou99]. Here, the computational costs are reduced by transforming the frequency samples into this Fourier representation to reuse these values for each point on the grid. A Hamming window is used to filter the frequency samples of one transmitter-receiver-pair. Afterwards, the inverse Fourier transformation followed by a frequency compensation and a baseband conversion are used to obtain a simpler reconstruction formula which is shown in Eq. 5.9.

$$C(x,y) = \sum_{tr=1}^{t\cdot r} A_{w,tr} \cdot e^{-i\varphi_{w,tr}} \cdot e^{i2\pi f_{cent,tr} \frac{d_{tr\to p(x,y)}}{c}} \tag{5.9}$$

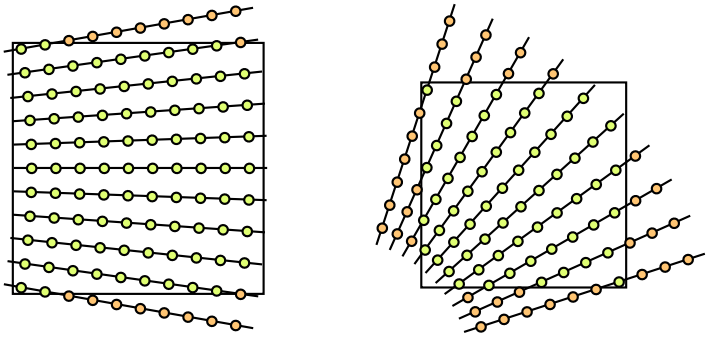
Performance gain by GPU implementation of reformulation

The intensity for each transmitter-receiver-combination is calculated by a single multiplication of the theoretical phase of the center frequency $f_{cent,tr}$ and the frequency sample $f_{w,tr}$. $f_{w,tr}$ is determined by interpolating the transformed frequency samples. While a straightforward GPU implementation needs only 2 seconds, a MATLAB implementation still needs about 6 minutes for the given example. The proposed GPU acceleration of Sec. 5.2.4.3, which improves the straightforward GPU implementation, leads to a processing time of 0.9 seconds for the same example.

5.2.4.3 *GPU Acceleration*

Precalculation of distances

The first GPU implementation calculates the distances between a transmitter, the reconstructed point of the slice and a receiver for each slice and each frame. First benchmarks revealed that this distance calculation is the main bottleneck because many square root operations are needed, e.g. an exemplary slice with 20 transmitters, 24 receivers and a resolution grid of 100 x 200 points requires 880000 square root operations. These operations can be reduced by reusing



(a) (b)

Figure 5.8: The geometrical positioning between view and bounding box influences the efficiency of precalculated distances. If the center of projection is far and aligned to the face of the bounding box, almost all points (green) can be used for reconstruction (a). Many points (orange) are not usable for reconstruction, if the center of projection is near and aligned to the edge of the bounding box (b). Image source: [PKK*13]

the calculated distances between consecutive slices of one view. The efficiency of that calculation is increased if lesser views and more slices per view are used. If the setup behavior does not change over time, a calculation of distances would only be required once for each view.

Geometrical unfolding allows to reuse the distances

The unfolding of the setup in the configuration step (see Sec. 5.2.2 and Sec. 5.2.3.3) allows a more efficient calculation of distances because the slice directions are interpreted as a rotation around a center of projection (cf. Fig. 5.2). Therefore, the same distances on each slice can be used for reconstructing points that lie in the bounding box. With the assumption that reconstructed points on consecutive slices keep similar distances to the bounding box, setups with parallel slices can benefit from precalculated distances as well.

Efficiency of precalculated distances

The position of the center of projection in relation to the position and rotation of the bounding box influences the spatial efficiency of precalculated distances. More points on the slices have to be calculated to cover the bounding box if the center of projection has a short distance to the bounding box and points to an edge of the bounding box. In comparison, almost all calculated points are usable if the center of projection is directing to a face of the bounding box and has a large distance to the bounding box. Examples for both cases with invalid and valid reconstruction points are shown in Fig. 5.8.

Distances are stored in GPU textures

A 3D texture with a resolution of the slice and individual layers for each transmitter and each receiver is used to store the precalculated distances. The distance values are accessed by cached texture-lookups. Since textures allow a faster read-/write-access in compari-

son to global memory access usually, they are used to store and load other intermediate results as well.

5.2.5 3D Image Generation

5.2.5.1 Related Work

Current THz visualizations base on raytracing of triangles

The reconstructed data is used to create a visual representation of the observed scene. Usually, commercial software is utilized for the rendering. However, solutions based on these commercial toolboxes show little performance ([GGLH06b, CV07, FDK07, YGZ09]). For generating the image, most solutions use raytracing techniques ([FDK07, MRBL02]). Therefore, the scene is represented by triangle meshes which are textured by images that hold additional material parameters like refraction indices etc. ([YGZ09, MRBL02]). If the traced rays found intersections with the scene geometry, the corresponding texture coordinates are used to obtain the correct material parameters by texture lookups. Those material values serve as basis for the image generation. The proposed visualization uses a raycasting to render a volume grid with interpolated reconstruction values of the discrete 2D slices.

[PG10] processes 2D slices for 3D visualization as well

[PG10] uses a similar approach regarding the acquisition and processing of the data on GPU. Here, discrete 2D slices of a mechanical movement are reconstructed and used as input for further processing in the visualization step as well. Furthermore, the slices are represented by textures too.

Multimodal Fusion

In addition to the volumetric THz representation, a visual 2D scene representation is obtained by an optical camera which is used as a second modality. While it is a common technique to fuse modalities in the context of depth cameras ([KTD*09, IKH*11, HKH*12]), the fusion of THz and visual representations is limited to 2D overlays of stylized scene representations or camera images (cf. [Tug13]). The proposed method creates a fused 3D representation by projecting the 2D camera image onto the volume grid.

5.2.5.2 Fusion of Multiple Views and Visualization

Overview

The proposed CUDA implementation fuses multiple views to one consistent 3D object, because several acquisition directions, i.e. views, allow to detect more surface details in comparison to an acquisition from only one direction. The implementation allows to interact with the reconstructed object in 3D. Specific visual properties like the reference value for db-scaling or the isothreshold of the isosurface rendering can be adjusted.

2D slices are fused

Two different approaches, which are discussed in the following, can be used to fuse multiple views. In this context, fusion refers to a combination of sets of reconstructed 2D slices. This fusion allows

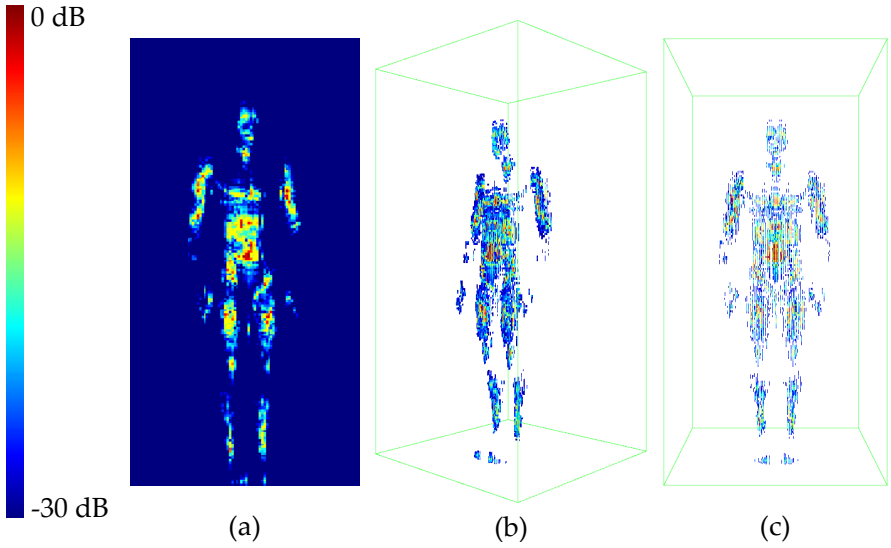


Figure 5.9: Alternative renderings for the single slices are shown. Stacked slices of one view can only be used for one acquisition direction (a). Correctly positioned slices of all views are placed in a 3D bounding box from two different viewing directions (b, c). In comparison to a viewing direction which differs from the acquisition direction (b), the visualization quality decreases if the direction of the rendering camera coincides with the acquisition direction of the scanner (c). Image source: [PKK*13]

the creation of a consistent 3D representation of the acquired object. A simple stacking of the slices (see Fig. 5.9 (a)) is not appropriate for a fusion of different acquisition views because the geometrically correct positioning of the slices is neglected.

Simple rendering of slices as planes

The first approach uses the 3D coordinates of all reconstructed slices to place them in the same bounding box. All slices of all views are positioned correctly so that the fused shape of a 3D object can be recognized. Examples for the correctly positioned slices can be seen in Fig. 5.9 (b, c). Every slice is represented by a rectangle, which is generated on the GPU. The rectangles are textured by images which contain the reconstructed intensity values of the respective slice. The more the viewing direction of the render camera coincides with the acquisition directions the less surfaces can be seen, i.e. a direct visualization of the slices leads to a view-dependent image quality.

Second approach adds slice intensities to a volume grid

Instead of creating geometrical representations of the slices, the second approach transforms the reconstructed intensity values of the slices into a common, view-independent Cartesian volume grid. This grid is defined by the bounding box and a resolution. The final intensity at each voxel of the grid is obtained by adding the separate intensities of all contributing views for that voxel. The intensity contribution of one view is interpolated from the neighboring slices of the respective voxel. By projecting the voxel center onto the surrounding

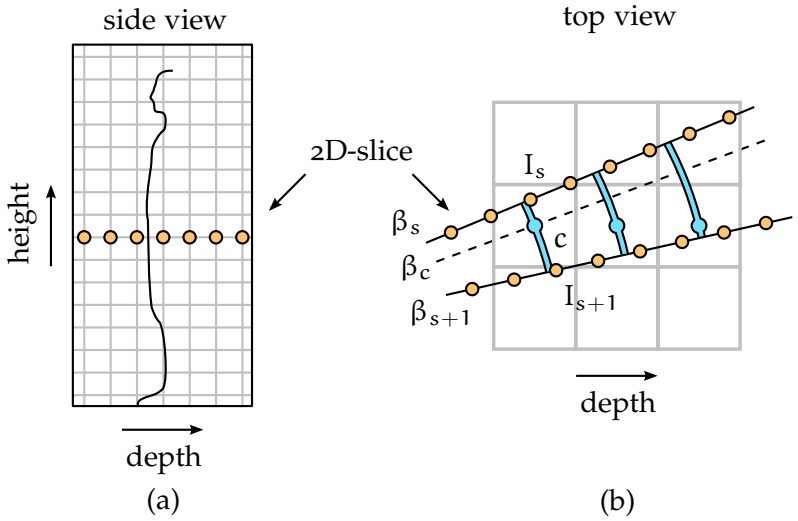


Figure 5.10: The used methods for interpolating the intensity values are depicted. For depth and height of the voxel center in the 2D slice, discrete values are bilinearly interpolated from texture lookups (a). A theoretical angle that intersects the voxel center is used to interpolate between these bilinearly interpolated intensity values of neighboring slices (b). Image source: [PKK*13]

Texture lookup provides linear interpolation in the slice

slices, bilinear interpolation within the slices and angular interpolation between the slices are used for compositing the intensity result.

The distance from the voxel center to the center of projection and the height of the voxel center determine the bilinear interpolation in the slice. These two values are used to create [u,v]-coordinates which allow to perform an interpolated texture lookup (see Fig. 5.10 (a)). The result is the reconstructed slice intensity I_s.

Angular interpolation between slices for lateral direction

The slice intensities I_s and I_{s+1}, which surround a voxel center c, are interpolated by an angular factor α. To calculate α by Eq. 5.10, a theoretical orientation angle β_c for a slice that intersects the voxel center is determined while each discrete slice is described by an orientation angle β_s. α serves as an interpolation factor between the neighboring slices to determine the intensity I_c at the voxel center (see Fig. 5.10 (b) and Eq. 5.11).

$$\alpha = \frac{\beta_c - \beta_s}{\beta_{s+1} - \beta_s} \tag{5.10}$$

$$I_c = (1 - \alpha) \cdot I_s + \alpha \cdot I_{s+1} \tag{5.11}$$

Interpolation in case of parallel slices

This interpolation with an angular factor can not be used for setups with parallel slices. Here, the interpolation factor α needs to be calculated by shortest distances d_s and d_{s+1} between the theoretical

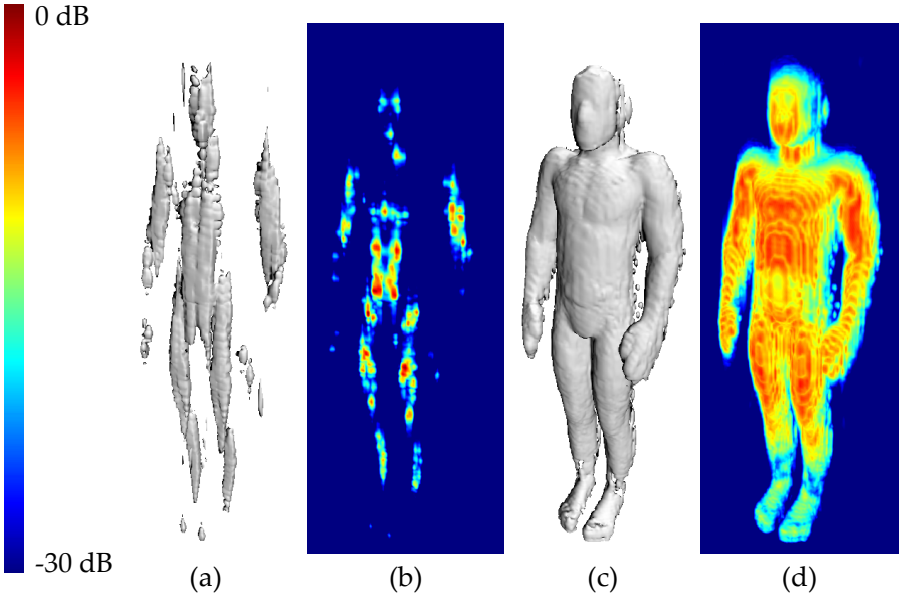


Figure 5.11: The implemented raycaster shows maximum intensity projection (b, d) and locally illuminated isosurfaces (a, c). Datasets from the simulation of [Kli12] with physically based attenuation factors (a, b) and without scattering (c, d) are rendered. The reference values of c and d are adjusted for visualization because ideal reflections result in much higher values. Image source: [PKK*13]

slice, which intersects the voxel center, and the neighboring slices (see Eq. 5.12).

$$\alpha = \frac{d_s}{d_{s+1} + d_s} \tag{5.12}$$

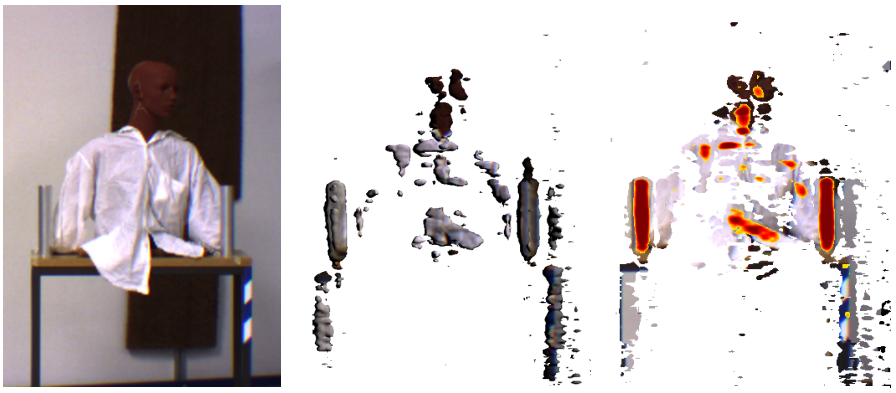
5.2.5.3 *Volume Raycasting with Multimodal Overlay*

Isosurfaces and maximum intensities are used

A raycaster based on methods of [EHK*06] is implemented to visualize the interpolated intensity values of the volume grid. Depending on an isothreshold, it is possible to render isosurface with local illumination. Furthermore, a maximum intensity projection can be shown as well. Fig. 5.11 shows examples for both types of visualization.

Correct texture coordinates for overlay rendering required

Additionally, it is possible to project a visual 2D scene representation onto the fused volume grid. This representation is obtained by an optical camera and is provided as a texture in the volume raycaster. Depending on the rendering technique, the maximum intensity or an isothreshold determine the point p which needs to be colored by the correct pixel value from the texture. To obtain this value, the correct texture lookup needs to be calculated by transforming p to texture coordinates. It requires a geometrical transformation from p to p' between bounding box and optical camera which is described in Sec. 5.2.3.4.



(a) (b) (c)

Figure 5.12: A test scene (a) is rendered with an overlay. Locally illuminated isosurfaces (b) and a combined rendering with maximum intensity projection of high intensities (c) are shown.

Perspective projection and coordinate normalization

Afterwards, a transformation of p' to 2D texture coordinates $[u, v]$ needs to be done. It consists of a projection and a normalization of 2D coordinates. To take into account the perspective projection of the optical camera, Eq. 5.13 is applied to obtain $[u', v']$ from p' which serves as input. The obtained range of the values is defined by the image width w and image height h. Therefore, a normalization by the texture size is needed to obtain $[u, v]$-coordinates in the usual range $[0, 1]$ (see Eq. 5.14). After a texture lookup with $[u, v]$, the obtained color is applied to the corresponding pixel in the final image.

$$[u', v'] = [p'_x/p'_z, p'_y/p'_z] \tag{5.13}$$

$$[u, v] = [u'/w, v'/h] \tag{5.14}$$

Multimodal overlay and rendering of high intensities

For the application of a body scanner, the rendering of the raw volume grid is combined with the overlay rendering. While the raw rendering shows suspicious regions, the overlay protects the privacy of the screened person. This rendering behavior is implemented by a user-defined threshold. If the intensity is higher than the threshold, the THz data is rendered by maximum intensity projection. If the intensity is below the threshold, the overlay rendering is used. Fig. 5.12 shows examples for a pure overlay rendering and a combined overlay rendering.

5.3 PIXELWISE SCANNING SYSTEM

5.3.1 *System*

Description of scanning system

In comparison to the hybrid scanner of Sec. 5.2, the complexity of the pixelwise scanning system is strongly reduced. It consists of a metallic frame with a THz transceiver and a THz receiver. These

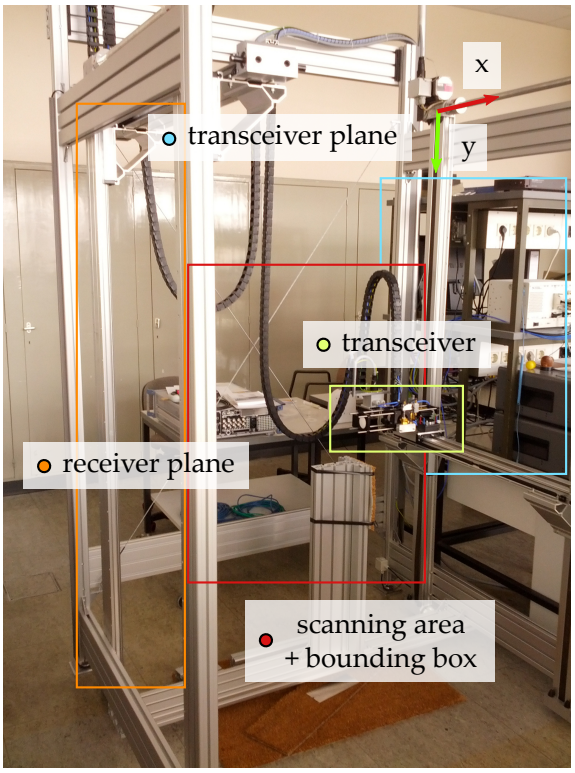


Figure 5.13: A photo of the pixelwise scanning system is shown. The metallic frame is used for mounting tracks which are used for moving receiver and transceiver in opposite planes along x- and y-direction. In between, the scanned objects are placed in the scanning area. The left receiver and a stage were not mounted while taking the photo.

two elements are mounted on opposite planes of the frame and can be moved mechanically in this plane, so that each position of the transceiver can be interpreted as a pixel of a 2D THz image of the object inside the frame. Therefore, the imaging is done by a sequential positioning of the THz transceiver and THz receiver. It allows to reconstruct the scattering behavior of the object in reflection and transmission mode. A photo of the scanning system is shown in Fig. 5.13.

Most of discussed methods of Sec. 5.2 are not applicable

The proposed methods of a geometrical calibration, an efficient GPU reconstruction or a multimodal overlay of the hybrid scanner are not necessary or can not be applied to the pixelwise scanner, because the imaging is done by only one transceiver and one receiver. Since both elements are moved mechanically, a parallel processing of measurements is not meaningful. Furthermore, an optical camera is not used and a visual scene representation is not created. A geometrical calibration is not needed as well, because the frame of the system defines the bounding box already and all required spatial positions are known.

Restrictions for applying the fusion of Sec. 5.2.5.2

If a rotation stage would be mounted in the center of the bounding box, several views of an object can be created and a fusion would be necessary. Unfortunately, the performance of the system is limited by this mechanical movement of the transceiver and a creation of several views would be time-consuming. Additionally, the object is limited in size and should be static during the measurement of one view. Therefore, a fusion of views would be possible, but not meaningful due to these restrictions of the data acquisition.

Configuration for an efficient simulation is possible

Although the discussed methods of Sec. 5.2 are not applicable for this scanner and processing of THz data is without GPU-acceleration, CG methods can be used to configure the system for the efficient simulation of the system. Sec. 5.3.2 discusses this configuration.

5.3.2 *Configuration*

Each measurement requires the antenna position

The simplicity of the geometrical properties of the setup and the reduced number of THz antennas exhibit the basic geometrical requirements for reconstructing 3D objects with THz imaging systems. Since each pixel is created by an individual position of the transceiver, it becomes clear that the spatial information of transmitter and receiver is required for each measurement in the general case.

Efficient configuration by describing the antenna movement

In a naive approach, every measurement is described explicitly by the geometrical position of a transmitter and a receiver. For a complex imaging with sophisticated scattering reconstruction, which combines several positions or more antennas per reconstructed point, this approach is very inefficient. Therefore, the degree of freedom for each THz antenna and the scanned object needs to be analyzed for an efficient geometrical formulation of the setup. While the most flexible approach would be a spatial path description of each antenna, the movements of the antennas are usually constrained and easier descriptions are obtainable.

Linear movement of the scanner can be described efficiently

In case of the scanning system, the THz antennas are moving linearly in a plane. Furthermore, the movement is aligned to the directions of Cartesian coordinate axes and offsets between pixels are constant. Therefore, a start position of the antenna S_{xy} for creating a pixel in the corner of the final image a constant offset o_c are sufficient to provide all antenna positions $a(i,j)$ by Eq. 5.15. The pixel indices i and j are used for addressing the x- and y-positions of the antenna, respectively.

$$a(i,j) = (S_x + o_c \cdot i, S_y + o_c \cdot j) \tag{5.15}$$

GPU can be used to simulate all pixels in parallel

This efficiency can not be exploited in the processing of the acquired data, because the transceiver is moving sequentially to all pixel positions, but the parallelism of the GPU can drastically speed up a simulation of the scanning system. Since the combination of all antenna positions leads to a reconstruction of a 2D image and parallel

pixel processing is one of the main tasks of GPGPU, the GPU can be exploited to calculate all contributions at all antenna positions in the same time by the simplified geometrical description.

5.4 SUMMARY

Differences between prototypes lead to individual solutions

Large differences in the setup complexity between the two prototypical scanner setups show that a generalized application of CG methods to all possible scanner setups is difficult to obtain. While only a configuration of the pixelwise system is used for an efficient parallelized simulation on GPU, several proposed CG methods have been applied to the hybrid scanning system.

Efficient system description is proposed by the configuration

Here, an efficient configuration is proposed to describe the complex geometrical behavior in the imaging by introducing an unfolding of geometrical THz radiation paths which led to an efficient interpretation of views on the scanned objects. Additionally, general reconstruction parameters and geometrical information of the bounding box are maintained.

Geometrical calibration allows a sophisticated visualization

To use this configuration for further processing and to allow a multimodal imaging, a geometrical calibration is done. First, a transformation between known reference points of the scanner and points of a freely placeable bounding box is calculated. This transformation is required for a fusion of the generated THz datasets. Second, an additional visual calibration allows an overlay of the THz reconstruction with an visual representation of the scene. This overlay is enabled by determining the transformation of bounding box coordinates to coordinates of the optical camera.

GPU accelerates reconstruction and fusion of 2D profiles

The GPU is applied to accelerate the reconstruction of discrete measurements of the scanner, which is done by synthetic aperture imaging techniques. The GPU allows a real-time reconstruction of 2D profiles from the screened object. To obtain a 3D representation, an additional step on GPU fuses those profiles to a consistent volume representation of the scene.

Volume raycaster with multimodal fusion is provided

To render the fused object, a volume raycaster is implemented. The two common techniques of isosurface rendering and maximum intensity projection are used. Additionally, the information from the visual calibration and the image of the optical camera allow a multimodal overlay. After obtaining the correct position in the volume grid, it is possible to render the raw information of the THz data at this position or the correct color value from the projected camera image depending on a user-defined threshold. This rendering leads to a performant and flexible visualization of the screened object and it secures the privacy in the case of body scanners.

Potential improvements by CG methods depend on the scanner setup

The creation of a configuration is the only common method for the two differing scanner setups, but it is obvious that the properties are difficult to generalize if an efficient system description is needed.

While the hybrid scanner greatly benefits from the introduced CG methods in the whole processing workflow, only a simulation of the pixelwise scanning system benefits from applicable CG techniques. It leads to the conclusion that CG methods are more meaningful and more beneficial if the scanner has a high geometrical complexity or if heavy computations are required for the imaging.

6 CONCLUSION

6.1 SUMMARY

Improving methods for THz imaging is the goal

In the present thesis, CG methods are applied to the domain of THz imaging. While the simulation of THz radiation is one aspect, the processing of measured data from prototypical scanning systems is another aspect. With the goal of improving efficiency and performance of current techniques in these two fields, several CG methods have been developed and have been used.

Voxels are the optimal choice for THz scene requirements

For the simulation of specific THz properties, an efficient voxel data structure is proposed. In comparison to triangle meshes which store surfaces only and volume models which store the complete inner structure, a voxel tree is a good compromise. It stores thin material layers with inner structures and keeps memory consumption low by efficient empty space representations. Additionally, voxels can have arbitrary material properties for simulating scattering behavior of highly detailed object representations.

Efficient voxelization and rendering of SVTs has been proposed

Therefore, the idea of efficient sparse voxel octrees (SVO) of [LK10] has been generalized to sparse voxel trees (SVT) in this thesis. The SVT needs less memory for storing the hierarchical information of the tree, because a parent node stores up to 125 child nodes. Although, the rendering of SVOs is faster than the rendering of SVTs due to the generalization aspect, a fast and efficient rendering of SVTs has been developed. Since a triangle mesh needs to be used as input for the creation of a highly detailed scene representation, a fast and sophisticated voxelization method has been developed. It allows a fast out-of-core SVT creation from massive triangle data on GPU.

Two THz simulations with a varying feature set are proposed

For the simulation of THz radiation, two approaches with a differing feature set have been proposed. The first simulation of [Kli12] concentrates on single-bounce reflections at isosurfaces, which are determined in a common volume rendering. Here, polarization with Frenet frames and Jones vectors, an antenna distribution based on the model of [TSI*06] and a statistical roughness scattering are considered. The second simulation exploits the implicit properties of the SVT which allows the calculation of penetration depths of materials and the roughness scattering at explicitly modeled surface details. An antenna distribution based on arbitrary blurring kernels and a focusing lens system are implemented. Furthermore, it is shown that a multi-bounce simulation might be feasible. Both simulation approaches show an improvement in performance for the calculation of specific THz effects. In addition, it can be concluded that the com-

bination of both methods is very complex. Partially, the combination of both simulations is not even meaningful because the performance would drop again while additional effects would not significantly influence the imaging properties of the respective THz scanner.

CG methods improve the processing of real THz data

For processing measured THz data of a prototypical scanner, general purpose approaches on GPU (GPGPU) have been developed. An existing backprojection algorithm has been transformed to allow a parallel and near real-time reconstruction of massive input data. Furthermore, a geometrical configuration is proposed. It enables an efficient creation of a 3D representation from discrete 2D datasets. Additionally, a geometrical calibration has been used to combine the outputs of visual and THz camera to a consistent 3D volume with an overlay.

THz processes are improved by exploitation of GPU and SVTs

All approaches of this thesis show that the application of CG methods can support and improve current THz processes which are related to prototypical imaging. Mainly, the use and transformation of methods on GPU lead to a performance gain or a more efficient task handling. Furthermore, the SVT is a promising data structure, which enables an efficient simulation even if the physical effects become more complex for the THz range.

6.2 FUTURE WORK

Further ideas are provided

Although the proposed approaches and methods show an improvement of current THz workflows already, the basic CG concepts of this thesis are still very promising for future work. The following subsections contain a discussion on rough ideas that may improve THz simulation in the future.

6.2.1 *Adaptive and Palette-Based SVTs*

Palette-based SVTs need less memory

Highly detailed SVTs have a high memory consumption due to high voxel counts with individual properties. The use of a dynamic storage of material attributes would reduce the required memory drastically. While the presented SVT implementation reduces the needed memory by compressing the attributes already, a palette-based approach like [DKB*16], would improve the memory footprint even further. Here, only the indices to a predefined color library are stored as material attributes, so that less bits are required for representing the color per voxel.

Adaptivity of SVT can be exploited by a reduced palette

If a palette is used to store materials, the adaptivity of the SVT becomes important for further memory reductions, because the probability of equal materials in all child nodes of one parent node increases. Therefore, an additional step for processing the created SVT needs to be introduced. In a bottom-up approach, all child nodes need to be compared. If all voxels of a parent node have the same

attributes, the child attributes can be removed and the parent just stores the material and the information that no additional attributes are given in finer hierarchy levels.

Spectral information could be stored

In the context of a THz simulation or spectral material representations in general, the index to a color can be replaced by an index to a detailed spectral footprint. In the context of linear spectral unmixing, it might even be meaningful to store scalar values, which are linked implicitly to available base spectra by the position in an indexed array. Here, the scalar values are multiplied by the respective spectra to get the influence of all available base spectra to a voxel.

Proposed ideas need a post-processing of the SVT

All these changes require a sophisticated attribute handling for the voxels to benefit from the reduced memory consumption. Since the indexing to a varying number of materials or colors influences the memory allocation for the attributes, the proposed out-of-core voxelization needs to be extended by a post-processing step if the out-of-core property should be kept. In the voxelization the necessary materials are determined and stored. With this information, the structure of material nodes can be adjusted and memory consumption can be reduced.

6.2.2 *Ray Queue for Multi-Bounce Simulations*

Ray segments should be queued for a multi-bounce simulation

The modified rendering of SVTs is applied to a THz simulation (see Sec. 4.5.2.2). Due to the radiation properties, a ray bending is implemented for a more realistic focusing behavior. Here, the focusing beam is approximated by linear ray segments, which are processed sequentially. This technique should be used for a correct multi-bounce simulation as well, because rays change the direction and may split to two or more rays, if they encounter a new material layer. Since only one ray can be used during the traversal, the other ray segments need to be stored in a *ray queue* to process all segments before new segments are added to the queue.

Maintaining individual ray contributions is not trivial

If such a queue is used for a multi-bounce simulation and the rays are altered after an intersection, the validity of arbitrary incidence angles is given (cf. Fig. 4.15). Since the rays depend on each other, the intersections and individual ray contributions need to be maintained during the traversal. While this behavior leads to a complex handling of intermediate results already, it will get even more complex if additional physical effects like polarization need to be simulated. Therefore, a handling of these intermediate ray contributions is left for future work.

6.2.3 *SVTs without Triangles and Rays*

Incomplete simulation due to rays and triangles

The most challenging and most complex task for future work is an even more correct simulation of THz radiation with high-

performance. Since the SVT creation is still based on triangle meshes and SVT rendering is based on rays, physical effects like diffraction or roughness scattering are still simulated incompletely.

Creation of inner structures by triangles can lead to errors

The problem of creating SVTs from triangles lies in the representation of inner structures. Since triangles represent outer hulls only, the information of inner structures needs to be created artificially by additional triangle layers as it is described in Sec. 4.5.2.1. Here, the normal and material attributes of a voxel rely on the number of intersected triangle attributes, so that mixed materials or wrong normals may influence the correctness. Some ideas for solving this problem are:

- a flood filling if closed surfaces are still represented by triangles
- an interactive creation of voxels without triangles
- reconstruction from sensor data which are stored in a point cloud or a grid representation already

Direct voxel creation with a guided flood filling is promising

The interactive creation of voxel representations at a high resolution may become a tedious task and the reconstruction from sensor data may lead to further problems with consistent surfaces. Therefore, a mix of an interactive voxel creation and a flood filling approach for creating inner structures seems to be the most promising approach. While material surfaces can be defined by the user, a guided flood filling can set the correct voxel properties inside the material.

Rendering may be replaced by a voxelization of wave geometries

Since physical wave effects are not covered by a ray-based rendering which is based on approximations of geometrical optics, the SVT rendering should be improved to achieve a more realistic simulation in the THz domain. An idea for a more correct approach is an interactive voxelization of geometries which explicitly model a wave, so that radiation and scene are in the same voxel representation. If the wave is propagated, the voxelization is executed iteratively until intersections with the scene geometry are encountered. Starting from these intersection points, additional wave geometries are generated and parts of the original wave geometry do not need to be tracked further. Many open problems arise from this idea, but interesting research approaches may be the result if these are solved.

BIBLIOGRAPHY

[Ala04] Alabaster C. M.: *The microwave properties of tissue and other lossy dielectrics*. PhD thesis, Cranfield university, 2004. (Cited on page 111.)

[AMHH08] Akenine-Möller T., Haines E., Hoffman N.: *Real-time Rendering*. Taylor & Francis Ltd., 2008. (Cited on page 13.)

[Arm12] Armstrong C. M.: The truth about terahertz - no other region of the electromagnetic spectrum has generated so much interest in recent years, but breathless hype still can't overcome the fundamental limits of physics. *IEEE Spectrum 49*, 9 (2012), 28. (Cited on page 22.)

[ÁSK14] Áfra A. T., Szirmay-Kalos L.: Stackless multi-bvh traversal for cpu, mic and gpu ray tracing. *Computer Graphics Forum 33*, 1 (2014), 129–140. (Cited on page 84.)

[ASS11] Ahmed S. S., Schiessl A., Schmidt L.-P.: A novel fully electronic active real-time imager based on a planar multistatic sparse array. *IEEE Transactions on Microwave Theory Techniques 59*, 12 (2011), 3567–3576. (Cited on page 145.)

[AW87] Amanatides J., Woo A.: A fast voxel traversal algorithm for ray tracing. In *Proceedings of Eurographics* (August 1987), EG '87, pp. 3–10. (Cited on pages vii, 66, 70, 74, and 75.)

[AZ07] Abbott D., Zhang X.-C.: Special issue on t-ray imaging, sensing, and retection. *Proceedings of the IEEE 95*, 8 (2007), 1509–1513. (Cited on page 19.)

[BD02] Benson D., Davis J.: Octree textures. *ACM Transactions on Graphics 21*, 3 (2002), 785–790. (Cited on page 30.)

[BHP15] Beyer J., Hadwiger M., Pfister H.: State-of-the-art in gpu-based large-scale volume visualization. *Computer Graphics Forum 34*, 8 (2015), 13–37. (Cited on page 66.)

[BLD13] Baert J., Lagae A., Dutré P.: Out-of-core construction of sparse voxel octrees. In *Proceedings of the 5th Conference on High-Performance Graphics* (July 2013), HPG '13, pp. 27–32. (Cited on pages 41, 42, and 52.)

[BLD14] Baert J., Lagae A., Dutré P.: Out-of-core construction of sparse voxel octrees. *Computer Graphics Forum 33*, 6 (2014), 220–227. (Cited on pages 41, 42, 49, 53, 54, 55, 57, 58, and 59.)

[BPK07] Bergström D., Powell J., Kaplan A. F. H.: A ray-tracing analysis of the absorption of light by smooth and rough metal surfaces. *Journal of Applied Physics 101*, 11 (2007), 113504. (Cited on pages 107 and 128.)

[BPK08] Bergström D., Powell J., Kaplan A. F. H.: The absorption of light by rough metal surfaces - a three-dimensional ray-tracing analysis. *Journal of Applied Physics 103*, 10 (2008), 103515. (Cited on pages 107 and 128.)

[Bre65] Bresenham J. E.: Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1 (1965), 25–30. (Cited on page 66.)

[BRGIG*14] Balsa Rodríguez M., Gobbetti E., Iglesias Guitián J., Makhinya M., Marton F., Pajarola R., Suter S. K.: State-of-the-art in compressed gpu-based direct volume rendering. *Computer Graphics Forum 33*, 6 (2014), 77–100. (Cited on page 66.)

[BS87] Beckmann P., Spizzichino A.: *The scattering of electromagnetic waves from rough surfaces*. Artech House, 1987. (Cited on page 106.)

[CB04] Christensen P. H., Batali D.: An irradiance atlas for global illumination in complex production scenes. In *Proceedings of the Eurographics Symposium on Rendering* (June 2004), EGSR '04, pp. 133–141. (Cited on page 30.)

[CDE*14] Cigolle Z. H., Donow S., Evangelakos D., Mara M., McGuire M., Meyer Q.: A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques 3*, 2 (2014), 1–30. (Cited on pages vii and 36.)

[CG12] Crassin C., Green S.: Octree-based sparse voxelization using the GPU hardware rasterizer. In *OpenGL Insights*, Cozzi P., Riccio C., (Eds.). Taylor & Francis Inc., 2012, pp. 303–319. (Cited on page 41.)

[Cha04] Chamberlain J. M.: Where optics meets electronics: Recent progress in decreasing the terahertz gap. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences 362*, 1815 (2004), 199–213. (Cited on page 19.)

[CHB*12] CUYPERS T., HABER T., BEKAERT P., OH S. B., RASKAR R.: Reflectance model for diffraction. *ACM Transactions on Graphics 31*, 5 (2012), 122:1–122:11. (Cited on page 13.)

[CJMS00] CHEW W. C., JIN J.-M., MICHIELSSEN E., SONG J.: *Fast and Efficient Algorithms in Computational Electromagnetics*. Artech House Publishers, 2000. (Cited on page 98.)

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (February 2009), I3D '09, pp. 15–22. (Cited on pages vii, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 37, 66, 67, and 95.)

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum 30*, 7 (2011), 1921–1930. (Cited on page 67.)

[CT82] COOK R. L., TORRANCE K. E.: A reflectance model for computer graphics. *ACM Transactions on Graphics 1*, 1 (1982), 7–24. (Cited on pages 15 and 16.)

[CV07] COCHERIL Y., VAUZELLE R.: A new ray-tracing based wave propagation model including rough surfaces scattering. *Progress in Electromagnetics Research 75* (2007), 357–381. (Cited on pages 99, 106, 107, 108, and 148.)

[Dav10] DAVIDSON D. B.: *Computational Electromagnetics for RF and Microwave Engineering*. Cambridge University Press, 2010. (Cited on page 98.)

[DBB06] DUTRÉ P., BALA K., BEKAERT P.: *Advanced Global Illumination*. Taylor & Francis Inc., 2006. (Cited on page 13.)

[DGPR02] DEBRY D. G., GIBBS J., PETTY D. D., ROBINS N.: Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics 21*, 3 (2002), 763–768. (Cited on page 30.)

[DKB*16] DADO B., KOL T. R., BAUSZAT P., THIERY J.-M., EISEMANN E.: Geometry and attribute compression for voxel scenes. *Computer Graphics Forum 35*, 2 (2016), 397–407. (Cited on pages 25, 30, 66, 96, and 158.)

[DKLB13] DING J., KAHL M., LOFFELD O., BOLÍVAR P. H.: Thz 3-d image formation using sar techniques: simulation, processing and experimental results. *IEEE Transactions on Terahertz Science and Technology 3*, 5 (2013), 606–616. (Cited on page 21.)

[DMBM05] De Maagt P., Bolívar P. H., Mann C.: *Terahertz Science, Engineering and Systems - from Space to Earth Applications*. John Wiley & Sons, Inc., 2005, pp. 5175–5194. (Cited on page 22.)

[DW15] Demir I., Westermann R.: Vector-to-closest-point octree for surface ray-casting. In *Vision, Modeling and Visualization* (October 2015), pp. 65–72. (Cited on page 67.)

[Ede06] Edelsbrunner H.: *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2006. (Cited on page 7.)

[EHK*06] Engel K., Hadwiger M., Kniss J. M., Rezk-Salama C., Weiskopf D.: *Real-Time Volume Graphics*. Taylor & Francis Ltd., 2006. (Cited on pages 12 and 151.)

[FDK07] Fetterman M. R., Dougherty J., Kiser W. L.: Scene simulation of millimeter-wave images. In *IEEE Antennas and Propagation Society International Symposium* (June 2007), pp. 1493–1496. (Cited on page 148.)

[FSH*05] Federici J. F., Schulkin B., Huang F., Gary D., Barat R., Oliveira F., Zimdars D.: Thz imaging and sensing for security applications - explosives, weapons and drugs. *Semiconductor Science and Technology 20*, 7 (2005), 266. (Cited on page 22.)

[FTI86] Fujimoto A., Tanaka T., Iwata K.: Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications 6*, 4 (1986), 16–26. (Cited on page 66.)

[FvSB*11] Friederich F., von Spiegel W., Bauer M., Meng F., Thomson M., Boppel S., Lisauskas A., Hils B., Krozer V., Keil A., Löffler T., Henneberger R., Huhn A. K., Spickermann G., Bolívar P. H., Roskos H. G.: Thz active imaging systems with real-time capabilities. *IEEE Transactions on Terahertz Science and Technology 1*, 1 (2011), 183–200. (Cited on pages 1 and 22.)

[GGLH06a] Grafulla-González B., Lebart K., Harvey A. R.: Physical optics modelling of millimetre-wave personnel scanners. *Pattern Recognition Letters 27*, 15 (2006), 1852–1862. (Cited on page 99.)

[GGLH06b] Grafulla-González B., Lebart K., Harvey A. R.: Physical optics modelling of millimetre-wave personnel scanners. *Pattern Recognition Letters 27*, 15 (2006), 1852 – 1862. (Cited on page 148.)

[GMM*02] GONZALO R., MARTINEZ B., MANN C. M., PELLEMANS H., BOLÍVAR P. H., DE MAAGT P.: A low-cost fabrication technique for symmetrical and asymmetrical layer-by-layer photonic crystals at submillimeter-wave frequencies. *IEEE Transactions on Microwave Theory and Techniques 50*, 10 (2002), 2384–2392. (Cited on pages 127, 130, and 132.)

[Hec02] HECHT E.: *Optics*. Addison Wesley, 2002. (Cited on page 99.)

[HKH*12] HENRY P., KRAININ M., HERBST E., REN X., FOX D.: Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research 31*, 5 (2012), 647–663. (Cited on page 148.)

[HN12] HEITZ E., NEYRET F.: Representing appearance and pre-filtering subpixel data in sparse voxel octrees. In *Proceedings of the 4th Conference on High-Performance Graphics* (June 2012), HPG '12, pp. 125–134. (Cited on page 67.)

[HSHH07] HORN D. R., SUGERMAN J., HOUSTON M., HANRAHAN P.: Interactive kd tree gpu raytracing. In *Proceedings of the Symposium on Interactive 3D Graphics and Games* (May 2007), I3D '07, pp. 167–174. (Cited on pages vii, 66, and 67.)

[Hun78] HUNTER G. M.: *Efficient Computation and Data Structures for Graphics*. PhD thesis, Princeton University, 1978. (Cited on page 30.)

[IKH*11] IZADI S., KIM D., HILLIGES O., MOLYNEAUX D., NEWCOMBE R., KOHLI P., SHOTTON J., HODGES S., FREEMAN D., DAVISON A., FITZGIBBON A.: Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *24th Annual ACM Symposium on User Interface Software and Technology* (October 2011), UIST '11, pp. 559–568. (Cited on page 148.)

[INH99] IOURCHA K., NAYAK K., HONG Z.: System and method for fixed-rate block-based image compression with inferred pixel values, September 1999. US Patent 5956431. (Cited on page 37.)

[ISO10] *ISO 4287:2010-07: Geometrical Product Specifications (GPS) - Surface texture: Profile method - Terms, definitions and surface texture parameters*. Standard, International Organization for Standardization, Geneva, October 2010. (Cited on page 102.)

[JT80] Jackins C. L., Tanimoto S. L.: Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing 14*, 3 (1980), 249 – 270. (Cited on page 30.)

[Kaj86] Kajiya J. T.: The rendering equation. *ACM SIGGRAPH Computer Graphics 20*, 4 (1986), 143–150. (Cited on pages 16 and 17.)

[KH12] Kirk D. B., Hwu W.-m. W.: *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 2012. (Cited on page 18.)

[KKP*12] Kahl M., Keil A., Peuser J., Löffler T., Pätzold M., Kolb A., Sprenger T., Hils B., Bolívar P. H.: Standoff real-time synthetic imaging at mm-wave frequencies. In *Proceedings of SPIE* (April 2012), Passive and Active Millimeter-Wave Imaging XV, p. 836208. (Cited on pages 2 and 136.)

[KLD*10] Krozer V., Löffler T., Dall J., Kusk A., Eichhorn F., Olsson R., Buron J., Jepsen P., Zhurbenko V., Jensen T.: Terahertz imaging systems with aperture synthesis techniques. *IEEE Transactions on Microwave Theory and Techniques 58*, 7 (2010), 2027–2039. (Cited on pages 21, 22, 139, and 145.)

[Kli12] Klinkert T.: *Simulation of a 3D THz imaging system based on first order surface elements*. Diploma thesis, University of Siegen, 2012. (Cited on pages ix, 104, 105, 107, 115, 123, 124, 128, 133, 151, and 157.)

[KSA13] Kämpe V., Sintorn E., Assarsson U.: High resolution sparse voxel DAGs. *ACM Transactions on Graphics 32*, 4 (2013), 101. (Cited on pages 25, 30, 41, and 66.)

[KTD*09] Kim Y. M., Theobalt C., Diebel J., Kosecka J., Miscusik B., Thrun S.: Multi-view image and tof sensor fusion for dense 3d reconstruction. In *IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)* (September 2009), pp. 1542–1549. (Cited on page 148.)

[Lai13] Laine S.: A topological approach to voxelization. In *Proceedings of the Eurographics Symposium on Rendering* (June 2013), EGSR '13, pp. 77–86. (Cited on page 40.)

[LCL89] Ling H., Chou R.-C., Lee S.-W.: Shooting and bouncing rays: Calculating the rcs of an arbitrarily shaped cavity. *IEEE Transactions on Antennas and Propagation 37*, 2 (1989), 194–205. (Cited on pages 98, 106, and 125.)

[Lee09] LEE Y.-S.: *Principles of Terahertz Science and Technology*. Springer, 2009. (Cited on pages 5, 19, and 20.)

[LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast BVH construction on GPUs. *Computer Graphics Forum 28*, 2 (2009), 375–384. (Cited on page 47.)

[LH91] LAUR D., HANRAHAN P.: Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (June 1991), SIGGRAPH '91, pp. 285–288. (Cited on page 30.)

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: Octree textures on the gpu. In *GPU Gems 2*. Addison-Wesley Longman, 2005, pp. 595–613. (Cited on pages vii, 30, 31, 32, 35, and 37.)

[LK10] LAINE S., KARRAS T.: Efficient sparse voxel octrees. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (February 2010), I3D '10, pp. 55–63. (Cited on pages vii, ix, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 37, 50, 65, 66, 67, 69, 70, 72, 73, 74, 77, 78, 80, 82, 85, 87, 88, 95, and 157.)

[LK11] LAINE S., KARRAS T.: Efficient sparse voxel octrees. *IEEE Transactions on Visualization and Computer Graphics 17*, 8 (2011), 1048–1059. (Cited on pages 27, 41, 53, 54, 55, 57, 58, and 59.)

[LW90] LEVOY M., WHITAKER R.: Gaze-directed volume rendering. In *Proceedings of the Symposium on Interactive 3D Graphics* (January 1990), I3D '90, pp. 217–223. (Cited on page 30.)

[Mea82] MEAGHER D.: Geometric modeling using octree encoding. *Computer graphics and image processing 19*, 2 (1982), 129–147. (Cited on page 30.)

[MMS*11] MALTSEV A., MASLENNIKOV R., SEVASTYANOV A., LOMAYEV A., KHORYAEV A.: Statistical channel model for 60 ghz wlan systems in conference room environment. *Radioengineering 20*, 2 (2011), 409 – 422. (Cited on page 110.)

[Mor66] MORTON G. M.: *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. IBM, 1966. (Cited on pages 34, 42, and 47.)

[MRBL02] MAMETSA H.-J., ROUAS F., BERGES A., LATGER J.: Imaging radar simulation in realistic environment using shooting

and bouncing rays technique. In *International Symposium on Remote Sensing* (February 2002), pp. 34–40. (Cited on page 148.)

[Mus13] Museth K.: Vdb: High-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics 32*, 3 (2013), 27. (Cited on page 30.)

[NIK91] Nayar S. K., Ikeuchi K., Kanade T.: Surface reflection: physical and geometrical perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 7 (1991), 611–634. (Cited on page 99.)

[PG10] Peinecke N., Groll E.: Integration of a 2.5d radar simulation in a sensor simulation suite. In *Proceedings of the 29th Digital Avionics Systems Conference (DASC)* (October 2010), pp. 3.A.6–1 – 3.A.6–9. (Cited on page 148.)

[PHJ16] Pharr M., Humphreys G., Jakob W.: *Physically Based Rendering*. Elsevier Ltd., 2016. (Cited on page 13.)

[Pho75] Phong B. T.: Illumination for computer generated pictures. *Communications of the ACM 18*, 6 (1975), 311–317. (Cited on page 13.)

[PJJK11] Priebe S., Jacob M., Jansen C., Kürner T.: Non-specular scattering modeling for thz propagation simulations. In *Proceedings of the 5th European Conference on Antennas and Propagation (EUCAP)* (April 2011), pp. 1–5. (Cited on page 99.)

[PJK11] Priebe S., Jacob M., Kürner T.: Polarization investigation of rough surface scattering for thz propagation modeling. In *Proceedings of the 5th European Conference on Antennas and Propagation (EUCAP)* (April 2011), pp. 24 – 28. (Cited on page 110.)

[PK15] Pätzold M., Kolb A.: Grid-free out-of-core voxelization to sparse voxel octrees on gpu. In *Proceedings of the 7th Conference on High-Performance Graphics* (August 2015), HPG '15, pp. 95–103. (Cited on pages 2, 40, 44, 45, 46, 48, 51, 52, 53, 54, 55, 56, 59, 60, 61, and 62.)

[PKK*13] Pätzold M., Kahl M., Klinkert T., Keil A., Löffler T., Bolívar P. H., Kolb A.: Simulation and data-processing framework for hybrid synthetic aperture thz systems including thz-scattering. *IEEE Transactions on Terahertz Science and Technology 3*, 5 (2013), 625–634. (Cited on pages 2, 105, 107, 108, 109, 110, 112, 113, 114, 135, 137, 138, 147, 149, 150, and 151.)

[Pre93] PREPARATA F. P.: *Computational Geometry: An Introduction*. Springer, 1993. (Cited on page 7.)

[RBI12] RYLANDER T., BONDESON A., INGELSTRÖM P.: *Computational Electromagnetics*. Springer, 2012. (Cited on page 98.)

[RDGK12] RITSCHEL T., DACHSBACHER C., GROSCH T., KAUTZ J.: The state of the art in interactive global illumination. *Computer Graphics Forum 31*, 1 (2012), 160–188. (Cited on page 1.)

[Sch94] SCHLICK C.: An inexpensive brdf model for physically-based rendering. *Computer Graphics Forum 13*, 3 (1994), 233–246. (Cited on page 15.)

[Sie02] SIEGEL P. H.: Terahertz technology. *IEEE Transactions on microwave theory and techniques 50*, 3 (2002), 910–928. (Cited on page 19.)

[SK10] SANDERS J., KANDROT E.: *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison Wesley, 2010. (Cited on page 18.)

[SM09] SHIRLEY P., MARSCHNER S.: *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., 2009. (Cited on pages 5 and 13.)

[SML*12] SADEGHI I., MUNOZ A., LAVEN P., JAROSZ W., SERON F., GUTIERREZ D., JENSEN H. W.: Physically-based simulation of rainbows. *ACM Transactions on Graphics 31*, 1 (2012), 3:1–3:12. (Cited on page 13.)

[Sou99] SOUMEKH M.: *Synthetic Aperture Radar Signal Processing with MATLAB Algorithms*. John Wiley and Sons, Inc., 1999. (Cited on page 146.)

[SS10] SCHWARZ M., SEIDEL H.-P.: Fast parallel surface and solid voxelization on GPUs. *ACM Transactions on Graphics 29*, 6 (2010), 179:1–179:10. (Cited on pages 40, 42, 47, 49, and 50.)

[ST07] SALEH B., TEICH M.: *Fundamentals of Photonics*. Wiley Series in Pure and Applied Optics. Wiley, 2007. (Cited on pages 1 and 99.)

[Ton07] TONOUCHI M.: Cutting-edge terahertz technology. *Nature photonics 1*, 2 (2007), 97–105. (Cited on page 19.)

[TSI*06] TOYODA I., SEKI T., IIGUSA K., SAWADA H., FUJITA Y., ORIHASHI N.: *Reference antenna model with side lobe*

for TG3c evaluation, IEEE doc. 802.15-06/474r0, November 2006. (Cited on pages 110, 124, and 157.)

[Tug13] Tugas J. F.: Privacy and body scanners at eu airports. *Novática Privacy and New Technologies*, S2013 (2013), 49–54. (Cited on page 148.)

[Vit01] Vitter J. S.: External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys 33*, 2 (2001), 209–271. (Cited on page 1.)

[vW06] van Waveren J.: Real-time dxt compression. 1–42. (Cited on page 37.)

[YFSPM12] Yu C., Fan S., Sun Y., Pickwell-MacPherson E.: The potential of terahertz imaging for cancer diagnosis: A review of investigations to date. *Quantitative imaging in medicine and surgery 2*, 1 (2012), 33. (Cited on page 22.)

[YGZ09] Yan L., Ge R., Zhong S.: A framework of passive millimeter-wave imaging simulation for typical ground scenes. In *Sixth International Symposium on Multispectral Image Processing and Pattern Recognition (MIPPR)* (October 2009), p. 749409. (Cited on page 148.)

[ZB11] Zhang Z., Buma T.: Terahertz impulse imaging with sparse arrays and adaptive reconstruction. *IEEE Journal of Selected Topics in Quantum Electronics 17*, 1 (2011), 169–176. (Cited on page 145.)

[ZGHG11] Zhou K., Gong M., Huang X., Guo B.: Data-parallel octrees for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 17*, 5 (2011), 669–681. (Cited on page 50.)