Technische Universität München
Forschungsinstitut caesar in Bonn

Interactive Point-Based Visualization for Medical Applications

Bartosz von Rymon Lipiński

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Hans-Joachim Bungartz

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Dr.h.c.mult. Karl-Heinz Hoffmann
2. Univ.-Prof. Dr. Rüdiger Westermann

Die Dissertation wurde am 26.04.2007 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.09.2007 angenommen.

Zusammenfassung

Interaktive 3D Visualisierung spielt eine fundamentale Rolle in vielen aktuellen medizinischen Anwendungen. Die effektive Nutzung von traditionellen Datenstrukturen und Renderingtechniken wird aufgrund von immer größer werdenden Datensätzen zunehmend zu einer unüberwindbaren Herausforderung. In dieser Arbeit wird daher die erste konsistente Lösung für eine rein punktbasierte medizinische Visualisierungskette vorgestellt.

Die entwickelten Methoden resultieren in einer kompakten Repräsentation von geometrischen Objekten und ermöglichen so nur die relevanten Datenbereiche zu betrachten. Alle virtuellen Modelle werden aus minimalen Punktprimitiven konstruiert, die nur Positions- und Materialattribute beinhalten. Damit wird es möglich die strikte Trennung zwischen Volumen und Oberflächen aufzuheben und einen universellen Modellierungsansatz zu verfolgen. Die lineare Natur von Punktearrays wird für den Entwurf von Datenstrukturen konsequent ausgenutzt, um eine schnelle Datenverarbeitung und beschleunigtes Rendering mit Grafikhardware zu erreichen. Darüber hinaus unterstützt die resultierende Einfachheit und Flexibilität der Punktdaten die Implementierung von dynamischen Geometriemanipulationen, z.B. für die Simulation von chirurgischen Schnitten.

Die wichtigsten Beiträge dieser Arbeit umfassen zwei grundlegende Bausteine moderner medizinischer Applikationen: die Visualisierungskette und adaptive Darstellungstechniken zum Erhalt von Interaktivität. Der erste Teil beinhaltet ein punktbasiertes Explorationssystem, z.B. für die Echtzeitextraktion von Isooberflächen und schwellwertbasierten 3D Modellen. Der damit verbundene Single-Pass-Renderer verwendet Deferred Shading und Funktionalität programmierbarer Grafikhardware. Der zweite Teil bezieht sich auf ein Progressive Refinement Framework, inklusive einer blickwinkelabhängigen Adaption der Punktedichte, sowie einem sequentiellem Algorithmus zur Steuerung des Detailgrades. Bei allen Techniken wurde auf die Verwendung von hierarchischen Informationen, zugunsten von einfachen Datenschnittstellen und direktem Zugriff auf die Geometrie, verzichtet.

Die integrierte Natur des punktbasierten Ansatzes weist ein Potential für solche Anwendungsbereiche auf, die auf eine nahtlose Visualisierungsinfrastruktur angewiesen sind. Ein Beispiel dafür ist der Operationssaal der Zukunft. Alle Techniken wurden in einer wiederverwendbaren und erweiterbaren Softwarebibliothek zusammengefasst, um deren weitere Verwendung in wissenschaftlichen und industriellen Projekten zu fördern.

Abstract

Interactive 3D visualization plays a fundamental role in many modern medical applications. Given the steady growth of dataset sizes, the effective utilization of traditional data structures and rendering techniques becomes increasingly an insurmountable challenge. Thus, this work presents the first consistent solution for a purely point-based medical visualization pipeline.

The proposed methods result in a compact representation of geometry, making it possible to consider just relevant data portions. Virtual models are built from a set of minimal point primitives, including only position and material attributes. This allows to break with the strict separation between volumes and surfaces, effecting in a universal modeling approach. The linear nature of point arrays is consequently utilized for data structure design in order to support fast processing and graphics-hardware-accelerated rendering. Additionally, the resulting simplicity and flexibility of the point data facilitate the implementation of dynamic changes in geometry, e.g. for simulating surgical fractions.

The main contributions of this work comprise two important building blocks of a state-of-the-art medical application: the visualization pipeline and adaptive display techniques for preserving interactivity. The first part includes a point-based data exploration system, e.g. for real-time isosurfacing and thresholding. The associated single-pass renderer is using deferred shading and modern features of programmable graphics hardware. The second part refers to a progressive refinement framework, including view-dependent adaptation of point densities and a purely sequential level-of-detail algorithm. All techniques circumvent the utilization of hierarchical information, effecting in simple data interfaces and direct access to the geometry.

The integrated nature of the point-based approach exhibits potential for areas of applications, which require a seamless and interconnected visualization workflow, like the operating room of the future. All techniques have been incorporated into a reusable and extendible software library, focusing on their utilization in scientific and industrial projects.

Acknowledgments

This work would have not been possible without the enduring support and seamless cooperation of the people involved. First, I would like to thank Prof. Dr. Dr.h.c.mult Karl-Heinz Hoffmann, Dr. Erwin Keeve and Prof. Dr. Rüdiger Westermann for their advice, the great degree of freedom I have received to work on various scientific topics and the opportunity to pursue my doctoral studies. I am also grateful to my colleagues and friends from the Surgical Systems Lab at the research center caesar for many inspiring discussions about computer graphics and medical visualization. In particular, I would like to thank Thomas Jansen for his careful proofreading of the first draft of this work and Nils Hanßen for his suggestions. My thanks go also to Dr. med. Timo Dreiseidler for the medical proofreading and Dr. med. Lutz Ritter for his input on computer-assisted surgery. Additionally, I would like to thank Alexander Winter for providing the screenshots of the distraction osteogenesis planning application.

I would also like to express my gratitude to the people of the Dental 3D project and the associated industrial cooperation partners for their inspirations and helpful discussions. My special thanks go to Dr. Joachim Hey for giving me the insight into the industrial development of medical applications. I am also very grateful to all the people that have made the many volume datasets public available.

Last but not least, I would like to thank my parents for their substantial motivation and support over all the years. I am very grateful to my fiancée Monika for her great love, tolerance and patience. My thanks go also to my sister for her very helpful spelling corrections. Thank you all for helping me over the challenging times and your everlasting trust, making it possible to accomplish my goals.

Data Sources

The *Head* and *Legs* datasets are courtesy of Prof. Dr. Dr. Hans-Florian Zeilhofer (Department of Maxillofacial and Reconstructive Surgery, Kantonspital Basel, Switzerland). The *Ventricles* dataset is courtesy of Prof. Dr. Hans Schild (Department of Radiology,

University of Bonn, Germany). The *RP Micro-CT* dataset is courtesy of Prof. Dr. Tille (Munich University of Applied Sciences, Germany) and Prof. Dr. Seitz (University of Rostock, Germany). The *Stagbeetle* dataset is courtesy of Prof. Dr. Eduard Gröller (Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria), Prof. Dr. Georg Glaeser (Institute for Art and Technology, Vienna University of Applied Arts, Austria) and Johannes Kastner (Wels College of Engineering, Austria). The *Dry Skull* datasets are courtesy of Dr. Joachim Hey (SiCAT GmbH, Bonn, Germany). The *Brain* dataset is courtesy of the University of North Carolina, USA. The *Visible Female* dataset is courtesy of the National Library of Medicine, USA.

The *Abdomen*, *Marschner-Lobb* and *Vessels* volume datasets are from Prof. Dr. Dirk Bartz's collection of real-life volume datasets available at `http://www.volvis.org`. They are courtesy of Michael Meißner (Viatronix Inc., North Carolina, USA), Prof. Dr. Steve Marschner (Department of Computer Science, Cornell University, USA) and Dr. Richard Lobb (Department of Computer Science, University of Auckland, New Zealand), respectively.

The *Stanford Bunny*, *Engine Block*, *Sheep Heart* and *Tooth* datasets are from Dr. Stefan Roettger's volume library, available at `http://www9.informatik.uni-erlangen.de`. They are courtesy of Dr. Terry Yoo (High Performance Computing and Communications, National Library of Medicine, USA), General Electric and Center for In-Vivo Microscopy at the Duke University, USA, respectively.

Contents

1 Introduction **1**

1.1 Medical Visualization Pipeline 2

1.1.1 Traditional Techniques 4

1.1.2 Challenges 7

1.2 Point-Based Approach 8

1.2.1 Representation and Interactive Rendering 9

1.2.2 Prospects 10

1.3 Overview 11

1.3.1 Requirements 12

1.3.2 Contributions 12

2 Related Work **15**

2.1 Data Structures and Rendering Algorithms 15

2.1.1 Surfaces 16

2.1.2 Volumes 20

2.2 Techniques for Preserving Interactivity 23

2.2.1 Surfaces 23

2.2.2 Volumes 27

2.3 Discussion 29

2.3.1 State-of-the-Art 29

2.3.2 Limitations 29

3 Point-Based Visualization Pipeline **33**

3.1 Exploration of Volumes 36

3.1.1 Data Structures 38

3.1.2 Preprocessing 42

3.1.3 Incremental Update 44

3.1.4 Experimental Results 45

3.2 Universal Rendering 50
3.2.1 Splatting 52
3.2.2 Shading 58
3.2.3 Experimental Results 64

4 Adaptive Display of Point Data **75**
4.1 Progressive Refinement Framework 76
4.1.1 Refinement Priority Estimation 80
4.1.2 Point Density Control 82
4.1.3 Smooth Refinement during Interaction 85
4.1.4 Experimental Results 86
4.2 Sequential Decomposition of Point Objects 89
4.2.1 Reordering Algorithms 90
4.2.2 Visualization with Adaptive Point Sizes 95
4.2.3 Experimental Results 96

5 Implementation and Examples **101**
5.1 Point-based Viewer Application 103
5.2 Progressive Cutting Tool 105
5.3 Mandibular Distraction Osteogenesis 109

6 Conclusions **113**
6.1 Summary 113
6.2 Discussion 114
6.3 Future Work 116

List of Figures

1.1 First X-ray image created by Wilhelm Röntgen. 2
1.2 Dental implant planning software. 4
1.3 Traditional volume slice views of a human head CT scan. 5
1.4 Traditional isosurface and composite volume rendering. 6
1.5 Low detail composite volume rendering. 7
1.6 Extensions to traditional visualization techniques. 7
1.7 High-level data flow chart of a typical medical visualization application. . . 8
1.8 2D representations of basic geometric primitives. 9
1.9 Illustration of point-based splatting and raycasting. 10
1.10 Polygonal isosurface model at different detail levels. 10
1.11 High-level data flow chart of a point-based visualization application. 11

3.1 Isosurface raycasting of a human head with low information content. 34
3.2 Point-based data-processing on different levels of memory. 35
3.3 Visualization of a mandible with volumetric information. 38
3.4 Discretized span-space. 39
3.5 Span-triangle data structure. 41
3.6 Incremental update of the span-triangle data structure. 45
3.7 Point-based *Visible Female* dataset during a priori exploration. 47
3.8 Point-based maximum-intensity-projection of the *Abdomen* dataset. 54
3.9 Illustration of point-based hybrid visualization using viewing rays. 56
3.10 Point-based hybrid visualizations with enabled shading. 56
3.11 Point-based MIP and DMIP visualizations of brain vessels. 57
3.12 Point-based STS-MIPs of a human leg dataset. 57
3.13 Splat information after first step of universal rendering. 58
3.14 Clip plane sequence of a tooth isosurface model. 60
3.15 Image-based normal estimation, including noise and dark band artifacts. . 61
3.16 2D representation of the normal estimation operator. 62

3.17 Different strategies for averaging of micronormals. 63
3.18 Selection of microtriangles 63
3.19 Point-based surface visualization with specular highlighting. 64
3.20 Close-up views of a point-based surface model. 66
3.21 Normal estimation in volume and image space. 67
3.22 Point-based slice view with surface contour. 70
3.23 Isosurface visualizations of *Head* and *Legs* datasets. 71
3.24 Image-based normal estimation techniques for a MRI brain dataset. 72
3.25 Full MIP volume rendering of *Head* and *Vessels*. 73
3.26 Renderings of extended surfaces using transfer functions and clip planes. . 74

4.1 Problems with simple timeout-based progressive refinement. 77
4.2 2D illustration of an occlusion cone. 81
4.3 Refinement order priorities. 81
4.4 Point density functions for different values of the shift parameter. 83
4.5 Large changes of the point density allocation during interaction. 85
4.6 Comparison of different strategies for point-based progressive refinement. . 88
4.7 Point density curves in relation to per-frame rendering progress. 89
4.8 Example of a sequential decomposition. 92
4.9 First three iterations of the deflating spheres algorithm. 93
4.10 First three iterations of the BSP-based algorithm. 95
4.11 Failure of the maximum distance check for the BSP-based algorithm. . . . 95
4.12 Smoother visual appearance as a positive side-effect of viewport scaling. . . 96
4.13 Point size offset curves computed during sequential decomposition. 98
4.14 Coarse LOD rendering of *Female* and *Abdomen* datasets. 100

5.1 Data-processing architecture. 102
5.2 Visualization architecture. 102
5.3 Relationships between point-based application examples. 103
5.4 Point-based isosurface of a large micro-CT RP model. 105
5.5 Point-based renderings of non-medical datasets. 106
5.6 Cutting of a point-based dataset with a spherical tool. 107
5.7 Illustration of the delay during progressive cutting. 108
5.8 Overview of the distraction osteogenesis procedure. 109
5.9 Interoral distractor device. 110
5.10 Workflow of the mandibular distraction osteogenesis planning system. . . . 111

List of Algorithms

1 Full point extraction from the span-triangle. 40
2 Incremental point extraction from the span-triangle. 45
3 High-level overview of the universal point rendering algorithm. 51
4 Splatting step based on refinement order and point density control. 85
5 Straight-forward computation of a sequential decomposition. 91
6 Computation of a sequential decomposition via deflating spheres. 93
7 BSP-based computation of a sequential decomposition. 94
8 Basic cutting procedure for point-based models. 107
9 Progressive cutting procedure for point-based models. 108

List of Tables

3.1 Volume datasets used for point-based exploration. 46
3.2 A priori generation of the point-based exploration data structure. 46
3.3 A posteriori generation of the point-based exploration data structure. . . . 48
3.4 Point-based isosurface extraction compared to Marching Cubes. 48
3.5 Extraction of the point grid from the span-triangle. 50
3.6 Point-based isosurface rendering compared traditional techniques. 65
3.7 Volume datasets used for point-based rendering. 68
3.8 Point-based full MIP rendering compared to traditional techniques. 68
3.9 Point-based rendering of extended surfaces. 69

4.1 Per-frame initialization times for visibility-based progressive refinement. . . 86
4.2 Rendering rates during progressive refinement with depth-based priority. . 88
4.3 Computation times for sequential decomposition. 97
4.4 Rendering of fat points for the first rendering pass during interaction. . . . 97
4.5 Rendering of fat points with an adaptive viewport resolution. 98

Chapter 1

Introduction

The medical field has been one of the most important application areas for scientific visualization. Starting with the digitalization of medical imaging in the 1970s there is now a variety of modalities available for the acquisition of structural data. The most common examples are computer tomography (CT) and magnet resonance imaging (MRI) [BKM00]. Over the last 30 years various visualization techniques have been developed in order to provide high-quality images and to support the medical expert in his daily work.

Having its origin in a hard-copy world of X-ray film images (figure 1.1)[1], the traditional approach for displaying interior anatomical structures is the 2D visualization of volume data as a stack of slices, sometimes called *multiplanar reformation*. Typically, radiologists explore the data using the so-called *cine mode* via sequential browsing of individual slices [Toe03]. Although being still the most popular technique in clinical practice, the huge amount of data that is produced by recent medical scanners makes it impossible to explore the patient model in the traditional 2D way. Current medical volumes can have more than 1000 high-resolution slices, requiring gigabytes of memory. The dimension of the information increase is shown clearly, if compared to the traditional display of hard-copy images in a light box. Typically, only 20 to 60 slices can be viewed at the same time. Furthermore, a two dimensional representation provides only a limited view of the real world, i.e. slicing through the volume makes following of branching structures, such as blood vessels, a hard task.

Forced by the advances in volume rendering and isosurface extraction since the 1980s, 3D visualization has gained more importance in the medical community. By supplying the radiologist with additional information from the third dimension, it facilitates the inspection of complex and abnormal anatomy. Moreover, the 3D view has reduced the demand for many invasive diagnostic procedures, e.g. diagnostic laparotomy and colonoscopy. Starting with non-interactive rendering techniques, today there is a clear trend towards interactive solutions. Only if real-time visualization is achievable, 3D techniques will be fully accepted by the medical community [PP03]. Virtual and augmented reality systems, telemedicine as well as minimally invasive surgery are other driving forces for the utilization of 3D visual-

[1] Figure 1.1 is courtesy of Radiology Centennial, Inc., USA

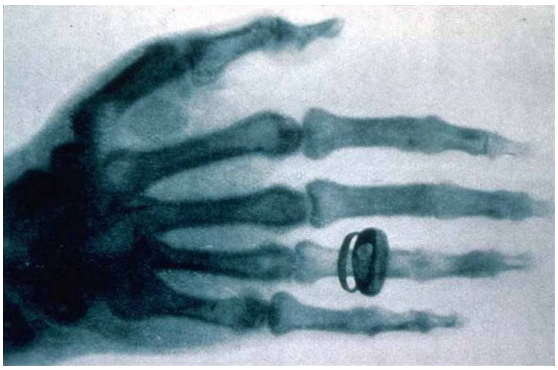


Figure 1.1: First X-ray image created by Wilhelm Röntgen at the end of the 19th century.

ization for interactive exploration of medical volume data. Nowadays, the increasing size of medical datasets and the implementation of complex patient-specific virtual environments impose new requirements for research and application development. Modern graphics hardware architectures, combined with smart algorithms for handling high-resolution datasets, are promising approaches to overcome the aforementioned challenges.

Several researchers have provided detailed surveys about the role of computer graphics in the medical domain, including Gross et al. and Bartz et al. [Gro98, BDH+04]. Additionally, Vidal et al. published in 2006 a recent report about medical visualization [VBB+06].

1.1 Medical Visualization Pipeline

The definition of the term "medical processing pipeline" is inspired by the idea that a typical medical workflow can be partitioned into a sequence of individual and closed algorithmic units, embedded in a data-processing framework. Additionally, a *medical visualization pipeline* takes advantage of techniques from computer graphics at different processing stages and creates visual representations of the data. Due to the diversity of clinical applications, there is not only one single pipeline that covers all potential aspects. Nevertheless, the following major building blocks of a typical medical visualization-related application can be identified [NMF99]:

Preprocessing can comprise the following steps: data acquisition using a medical scanner device, volume reconstruction from projection images, filtering techniques for noise reduction, segmentation, registration and creation of geometric models from volume data. Note that some data-processing steps may be optional and there is no specific order in this listing. For example, surface-based registration techniques require a prior construction of the surface model.

Exploration is used for the inspection of the generated virtual patient model mainly in diagnostic and educational applications [AUH+98]. This step can be implemented just by a pure rendering component, connected to the application's interaction system.

Nevertheless, additional visualization tools are usually incorporated, like placement of landmark points, distance and volume measurements and basic methods of data manipulation, e.g. setting of a cutting plane in order to get insight into the virtual object [HPP+95].

Simulation implies the manipulation of the underlying virtual patient model, e.g. using a virtual scalpel. This requires physically-based modeling and additional processing and visualization steps, such as decomposition of the object into a 3D mesh, association with physical properties, rendering of dynamic data etc. The most popular techniques in this context are the mass-spring model, finite element modeling (FEM) and particle systems [Gro99]. Simulation can be combined with a force-feedback device and plays an important role for training and surgical planning applications, like intra-operative deformation modeling.

Navigation is utilized for image-based guidance during a surgical procedure. Its main task is the display of the exact location of the real operating tool in relation to the virtual anatomical structures, used for the controlling of the surgical plan. Besides interactive rendering, navigation may include techniques from the preprocessing and simulation stages. Examples are real-time registration and modeling of tissue deformation during operation (e.g. brain shift). Sometimes, navigation refers also to the control of surgical robots.

Postprocessing corresponds mainly to application-specific steps that are performed "offline", i.e. typically after diagnosis or the actual surgical procedure. These steps include e.g. the validation of the surgical plan, scoring of results from a medical training procedure and generation of a diagnostic report. Other examples are rapid prototyping and manufacturing for the reproduction of missing bone parts, prosthesis design etc.

Interactive visualization has a central function in almost all steps of the pipeline. Surely, the most important role is exploration, but during the development of new data structures and rendering techniques for the medical domain the full pipeline has to be considered. One representative example, showing the mutual value of visualization, is segmentation. On the one hand, it can be an important preprocessing step for the creation of meaningful visualizations. On the other hand, visualization can also be an important factor for segmentation, e.g. for the validation of intermediate results and for potentially interactive algorithms, like thresholding.

Nowadays, many of the presented pipeline steps are incorporated in various medical areas of application. The most popular examples are maxillofacial, dental, orthopedic, neuro and liver surgery, virtual endoscopy and autopsy as well as radiotherapy [BDH+04] (figure 1.2)[2]. Moreover, it can be expected that their seamless and interconnected integration into the imaging, monitoring and information devices of the operating room of the future will play an increasingly important role [CCM04].

[2]Figure 1.2 is courtesy of Dr. Joachim Hey and Jochen Kusch (SiCAT GmbH, Bonn, Germany)

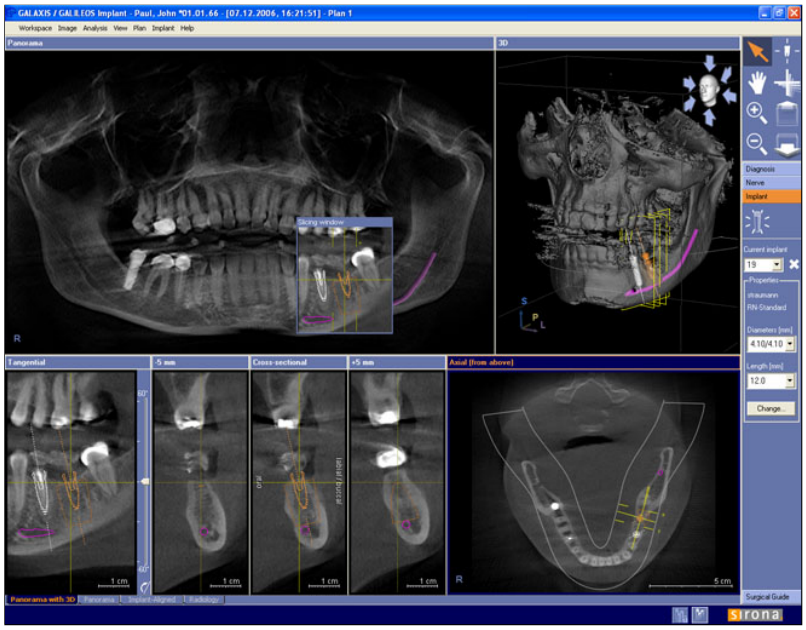


Figure 1.2: Dental implant planning software (GALILEOS$^{\text{Implant}}$ by SiCAT GmbH).

1.1.1 Traditional Techniques

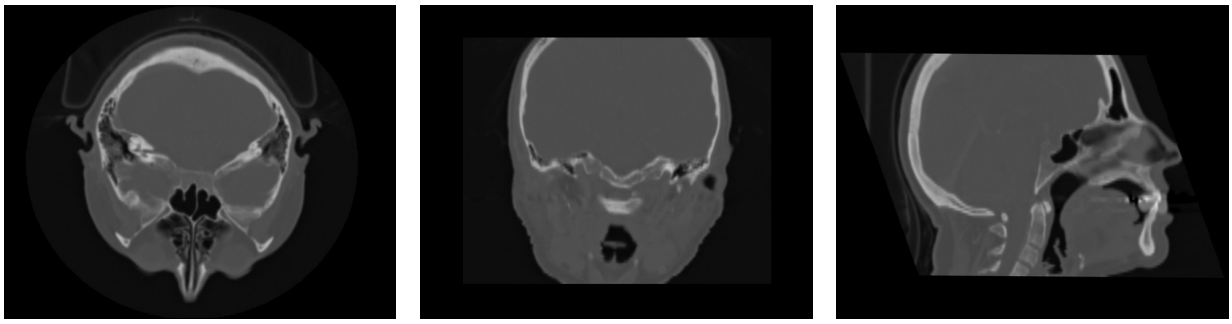
After the brief overview of the applicability of computer graphics in medicine, a more technical view of medical visualization will be described. In the following, the focus will be on the traditional visualization techniques, which are implemented in many state-of-the-art medical applications.

2D Visualization

The basic 2D medical visualization technique, called *slice view*, is implemented by sweeping through the 3D volume data with a plane. The orientation of the slice plane is usually axis-oriented, representing the classical transversal, sagittal and coronal views. Furthermore, arbitrary located slices are typically placed in regions-of-interest, i.e. by cutting through the currently examined anatomical structure.

Volumetric objects are mostly implemented as rigid blocks of memory, representing a regular 3D grid of sample values. Therefore, it is possible to incorporate fast incremental rasterization algorithms. The slicing can be performed in software or in hardware by taking advantage of 3D texture mapping functionality. The former approach provides much flexibility for the implementation of various extensions. The latter solution is very fast, but requires an upload of the volume data to graphics memory. An example, showing 2D slice views, is given in figure 1.3.

Alternatively, volumetric objects can be represented as 3D meshes of volume primitives, e.g. tetrahedra. Such data structures are common results of numerical simulation. Their irregular and highly interconnected nature makes the development of interactive visualiza-



(a) Transversal (b) Coronal (c) Sagittal

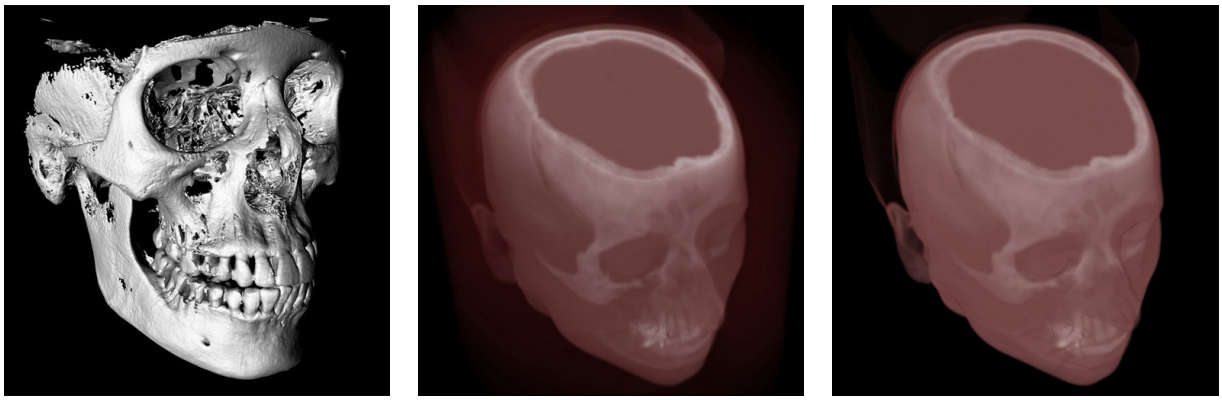
Figure 1.3: Traditional volume slice views of a human head CT scan.

tion techniques - even in case of 2D slicing - a challenging task. Similarly, applications that are working with surface-based models require the utilization of specific surface slicing techniques. Traditionally, surface models are represented with polygonal meshes, so the slicing procedure is realized with algorithms that have a close relation to polygonal cutting. For such cases, interactivity is preserved by incorporating acceleration techniques, e.g. based on a hierarchical bounding volume representation of the surface model.

3D Visualization

One important 3D technique for the medical domain is isosurface visualization. There are two different kinds of isosurfacing, *indirect* and *direct*. The former requires the execution of an extraction algorithm in order to generate a geometric model from the volume data. Since more than 15 years, the triangle-based *Marching Cubes* algorithm is the most popular approach [LC87]. Volume grid cells that are intersected by the isosurface are identified, polygonized and added to the output mesh. The resulting model is represented as a list of triangles and can be rendered efficiently with standard graphics hardware. A direct technique extracts the isosurface information in a view-dependent manner at rendering time utilizing ray intersections (figure 1.4(a)) or by applying transfer functions that correspond to a single spike in value space. Shading effects are computed on-the-fly by the corresponding 3D volume renderer [Lev88, SDWE98, WE98]. An explicit geometric representation is not constructed.

Isosurface visualization is typically used for CT-based volume data that represents large anatomical objects, such as bones and vascular structures. For other modalities, like MRI, it is often more useful to consider direct volume rendering (DVR), because similar anatomical structures may contain large scalar value variations. The most popular DVR techniques are raycasting and texture-mapping-based rendering (figures 1.4(b) and 1.4(c)). Volume raycasting is an image-based technique and is implemented traditionally in software, resulting in lower frame rates compared to graphics-hardware-accelerated approaches. Its advantages are high visual quality and flexibility on the implementation level [Lev90a, GBKG04a]. Recent activities provide also complete graphics-hardware-based



(a) Isosurface raycasting (b) Volume raycasting (c) 3D texture-mapping

Figure 1.4: Traditional isosurface and composite volume rendering.

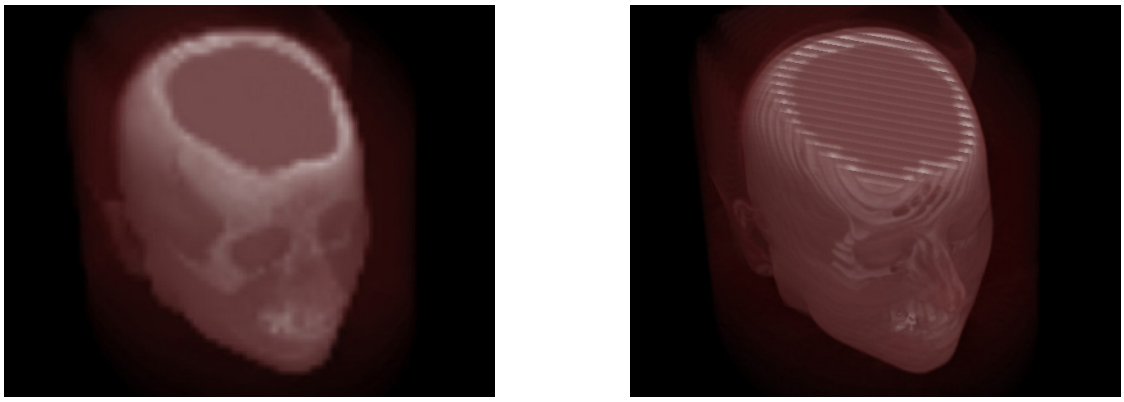
solutions [PBMH02, KW03, HSS+05]. Another strategy for DVR is based on volume slicing, combined with the utilization of 3D texture-mapping hardware [CN94]. The exploitation of hardware-based acceleration results in highly efficient rendering, freeing the CPU to perform other processing tasks. Similar to 2D slice viewing, such an approach requires the upload of the volume data to graphics memory.

Usually, a transfer function is applied to map data values of the rendered volume to optical properties, such as color and opacity. Compositing techniques that imitate the classical X-ray view are often of high interest for medical experts. An example is maximum intensity projection (MIP) rendering. Both approaches, raycasting and 3D-texture-mapping, offer straight-forward techniques for preserving interactivity by reducing temporarily the visual quality during interaction. As shown in figure 1.5, raycasting makes it possible to switch to a lower image resolution level and 3D-texture-mapping allows to reduce the number of texture slices, effecting in coarser samplings of the volume data. A comparison of popular DVR techniques, including shear-warp and volume splatting, has been presented by Meissner et al. [MHB+00]. Various splatting approaches are discussed also in section 2.1.2.

Extensions

Various extensions to the traditional visualization techniques have been developed, especially for the 3D case. This includes mainly the modification of existing rendering algorithms in order to emphasize specific anatomical structures and to overlay different rendering approaches in the same scene. Examples are hybrid rendering of volumes and isosurfaces as well as mixing of different volume rendering functions, e.g. transparency-based compositing and MIP [HMBG01, TIP05].

The most popular extensions are based on the incorporation of 2D slice information into the 3D view [PP03]. These approaches are particularly attractive for medical experts due to their direct correspondence to the 2D slice views and therefore intuitive user control. A



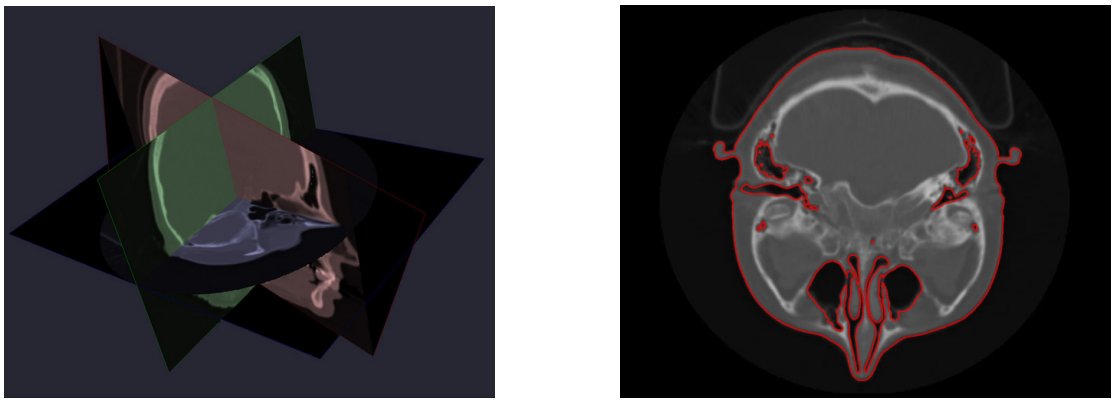
(a) Volume raycasting (b) 3D texture-mapping

Figure 1.5: Low detail composite volume rendering.

well-known tool is the *3D surface overlay plane*, i.e. the display of a cut 3D surface model together with one or more 3D slice planes. The 3D slices are rendered as textured quads, revealing the interior information of the associated volume (figure 1.6(a)). Their positions and orientations are linked to the settings of the corresponding 2D slice views. Similarly, the *2D slice contour* tool is used to display the contours, resulting from the slicing of the 3D surface model, together with the traditional 2D volume slice (figure 1.6(b)).

1.1.2 Challenges

The utilization of traditional techniques for the development of medical applications may come along with a set of demanding challenges. Perhaps the most crucial aspect is the steady growth of medical datasets as a result of the latest improvements in scanner technology and data-processing. In this context, one important argument of the point-based



(a) 3D slice planes (b) Slice with polygonal contour (red line)

Figure 1.6: Extensions to traditional visualization techniques.

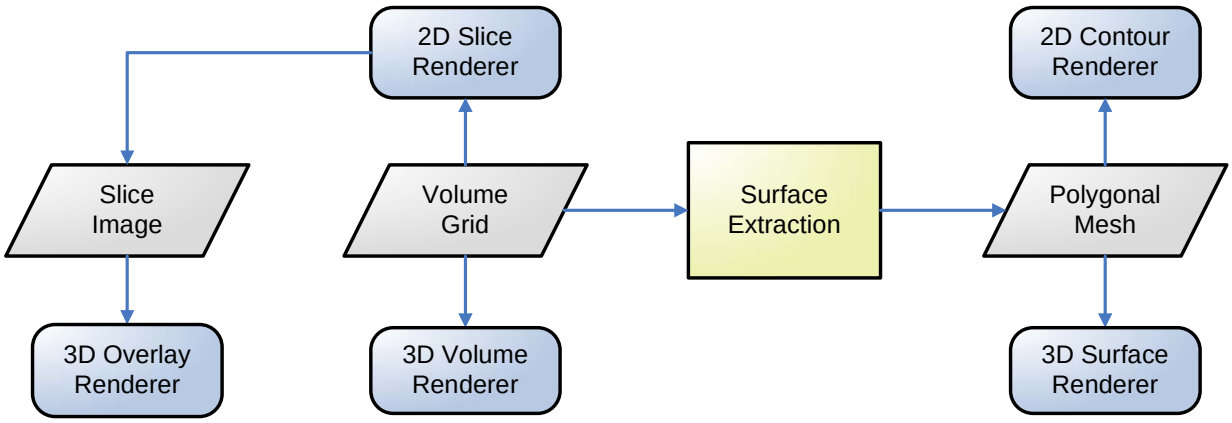


Figure 1.7: High-level data flow chart of a typical medical visualization application.

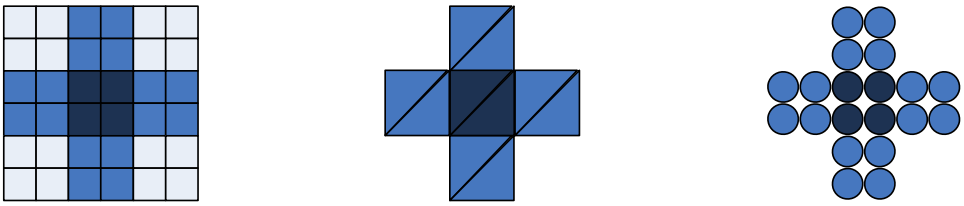
community is the fact that for many complex 3D models the projected size of the traditional primitives is smaller than the actual pixel size [RL00]. Typical drawbacks are unnecessarily high per-primitive costs and an outsized consumption of memory resources.

Most medical workflows are based on a heterogeneous object representation, including both: volume and surface data. On the one hand, surface models are popular due to their compact nature and hardware-supported visualization capabilities. On the other hand, volumetric representations are crucial for the full visual inspection of the virtual patient, volume measurements, heterogeneous tissue deformation etc. Using different and often complex data structures may lead to an overhead for users and developers. Examples are additional preprocessing times, delays in the workflow due to conversion between different data formats and development of extra rendering algorithms that can handle simultaneously both representations in the same view (figure 1.7).

Moreover, graphics-hardware-accelerated slice and volume renderers are limited by the currently available graphics memory. In the context of growing dataset sizes, this problem remains even for the latest generation of graphics hardware boards. Solutions are available, like bricking and smart memory management, but may effect in a reduction of the overall performance [LHJ99, WWH+00]. Using software-based rendering may also be an alternative solution, usually at the price of significantly smaller rendering rates.

1.2 Point-Based Approach

Converting volume data to an alternative primitive set and using it for the complete visualization pipeline may open new perspectives in the medical domain [Gro98]. In this work the utilization of point-based data for the implementation of medical workflows is described, resulting in a more unstructured representation for surface and volumetric models. Such an approach is a potential solution for many of the aforementioned challenges regarding a compact and flexible data representation, efficient rendering and preservation of interactivity.



(a) Pixels (b) Triangles (c) Points

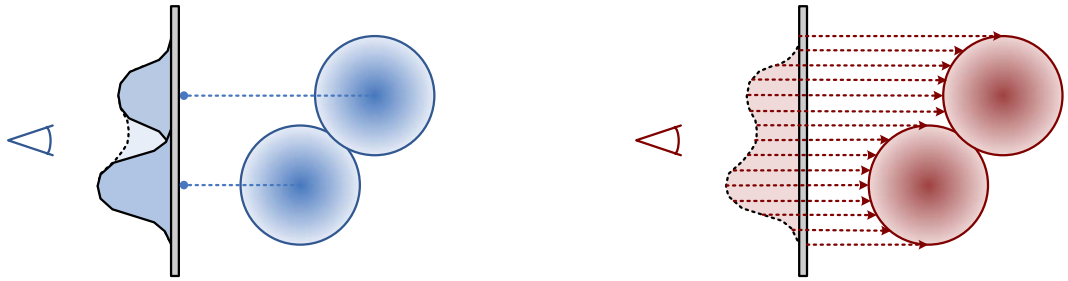
Figure 1.8: 2D representations of basic geometric primitives.

1.2.1 Representation and Interactive Rendering

A point-based object is defined by a set of points in geometric space, usually implemented as an array of independent data elements (figure 1.8). Each element contains positional information and optionally additional per-point attributes, such as a normal vector, material identifier, color etc. A point-based representation can be considered as a discrete sampling of the underlying geometry, without storing any connectivity and topological information. The neighborhood of a point is usually defined implicitly, e.g. using the nearest neighbor relation. In this context, traditional meshes can be seen as an "enhancement" of point sets [GPZ+04]. Although point data is the natural representation for 3D acquisition systems like laser range scanners, there are many different data sources, including the conversion of traditional data structures, sampling of implicitly or parametrically defined objects and simulation. Some researchers of the computer graphics community consider 3D points in analogy to 2D pixels. Others define 3D points as a nonuniform version of voxels.

Rendering of point data is sometimes interpreted as a resampling process, because the input is a list of nonuniform samples, whereas the output is considered as a set of samples on a regular grid [ZPvBG02]. The main challenge is the continuous reconstruction of the original shape and texture signals using local reconstruction kernels after warping the point object to the image plane. Beside traditional raycasting-based techniques, the most common approach for point-based rendering is *splatting*. It is an object-order rendering algorithm based on a consecutive projection of the 3D point primitives to image space, including a composition of the resulting *splat* contributions in 2D. Utilizing features of modern graphics hardware, such as programmable vertex and fragment stages, makes it possible to achieve fast rendering rates and high quality visual output [LKM01]. Figure 1.9 shows an illustration of the point-based splatting procedure compared to raycasting.

Another important aspect for the medical visualization pipeline is the preservation of interactivity during rendering and manipulation of the virtual patient model. A common solution to this problem is *progressive refinement*, having its origin in the networked image transmission and large dataset visualization [WGH83, LH91]. The main idea is to split the rendering process into multiple passes in order to give intermediate feedback to the user. This approach avoids the blocking of the application, preserving the interactivity of the whole system. Such a class of algorithms is usually called *output sensitive*, because they



(a) Splatting (b) Raycasting

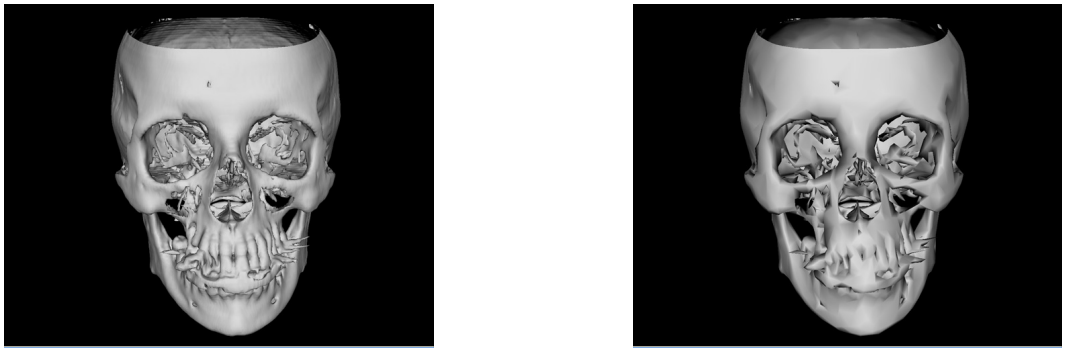
Figure 1.9: In point-based splatting, primitives are projected onto the image plane, including an accumulation of their contributions. In raycasting, sight rays are emitted from the image plane into the scene and tested for intersection with the primitives.

exhibit no or minimal dependency between the model and time complexity. Interactivity is achieved at the price of a coarse representation of the original model after the first rendering pass (figure 1.10). In order to ensure acceptable visual quality even for reduced data sizes, *multiresolution* data structures and *level-of-detail* (LOD) techniques play a substantial role in this context.

Please refer to sections 2.1 and 2.2, for more details on point-based rendering and techniques for preserving interactivity.

1.2.2 Prospects

Point primitives have a couple of advantages, making them interesting for the utilization in a medical visualization application. Many benefits result from the lack of explicit connectivity information in a point set, facilitating the design of memory efficient and flexible data structures [PGK02, Gro06]. Examples with high significance for the medical domain are multiresolution modeling and fast data manipulation techniques. Additionally, point



(a) Full detail level (b) Reduced detail level

Figure 1.10: Polygonal isosurface model at different detail levels.

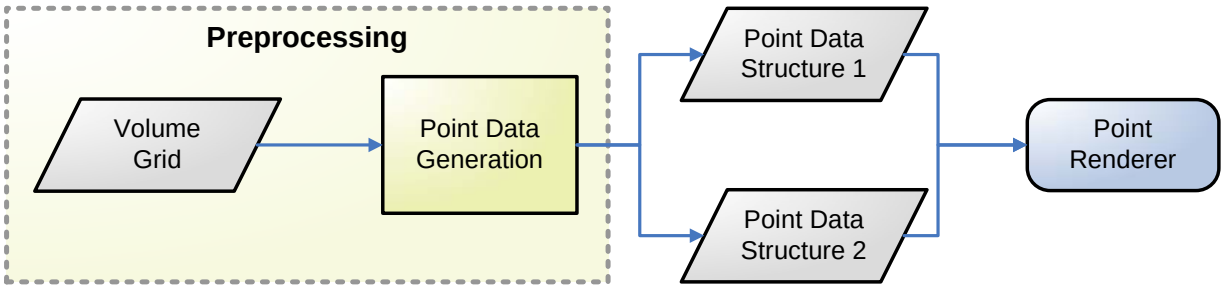


Figure 1.11: High-level data flow chart of a point-based visualization application.

elements that are arranged linearly in array-based data structures enable full applicability for direct processing on graphics hardware.

The growth of medical volumes implies an increase of less relevant regions in the dataset compared to areas of high interest. Examples are empty space, e.g. air surrounding the patient model, and less important anatomical structures that are out-of-focus for the current medical procedure. Therefore, representation of irregular regions gets more and more important. Traditional techniques try to solve this problem by developing dedicated *empty-space-skipping* or compression techniques [HSS+05]. Using points makes it possible to tackle the challenge in a more natural way. Scattered datasets can be represented compactly by placing samples only in the region of interest, defining an *irregular region-of-interest*.

There is a similar situation for surface-based representations. On the one hand, traditional triangle-based approaches are very efficient for large and flat regions of a virtual model. On the other hand, most medical datasets contain noise and highly detailed structures, preventing the exploitation of this feature. Moreover, highly interconnected data structures usually require the design of complex LOD management and progressive visualization techniques. In contrast, for many algorithms points can be interpreted as infinitesimal small and symmetric primitives, needing no local parametrization and orientation. The resulting simplicity of point-based algorithms can facilitate rapid development of medical applications, reducing the number of required software components and therewith the overall development costs (figure 1.11).

1.3 Overview

The development of the techniques that will be presented in this work has been inspired by the industrial focus of the associated research group. All point-based techniques have been encapsulated in a modular and reusable software library, called *PointPack*. In order to address both target users, researchers and application developers, the implementation has been based on a component-based software architecture. An important strategy has been the design of abstract and general interfaces in order to facilitate the extendibility of available visualization functionality.

1.3.1 Requirements

During the development of the 3D rendering architecture in this work, the main focus has been on a high degree of interactivity rather than very high visual quality. Nevertheless, it will be shown that it provides "adequate" results that are acceptable for many medical 3D visualization applications. Moreover, many solutions are based on complex hierarchical data structures and asymmetric point primitives. In order to achieve high rendering rates the design of plain data structures has been preferred, including a simple and maximally reduced point format.

Furthermore, full advantage of graphics-hardware-acceleration is taken, making it possible to activate additional computational power and avoid delays in the workflow [LKM01]. Interactivity is assured by utilizing a tight coupling between the data-processing components and GPU-based rendering. One example are sequential data structures that can be mapped directly to vertex arrays.

Another important requirement is the fast access to the point information. A typical medical workflow consists of many steps that include interaction with the virtual model, combined with modifications of the underlying data. Examples are sorting of rendering primitives, interactive segmentation, ablation and deformation operations. Consequently, it should be possible to modify the point data at all stages of the visualization pipeline. Isolated rendering techniques, such as image-based approaches, which require hours of preprocessing time to generate a dedicated rendering data structure, are not applicable [Wan04].

Finally, the design of the point-based progressive refinement and LOD techniques has been adapted to the specific nature of a typical medical scene. Usually, the 3D view contains only a few complex objects, representing the virtual patient model and a set of attached tool objects, such as landmark points, trajectory lines etc. Such a restricted scene depth prevents the utilization of many existing LOD solutions, which have been designed for large outdoor scenes or terrains. Please refer to sections 4.1 and 4.2 for more details.

1.3.2 Contributions

The major task of this work has been the tuning of available and the development of new point-based surface and volume visualization techniques for medical applications. The focus has been on efficient representation and interactive rendering of high-resolution volumetric datasets. The main contributions are as follows:

Point-based visualization pipeline - Various point-based data structures and algorithmic components have been designed in order to cover the main visualization-related steps of the medical workflow. An important aspect is the *span-triangle* data structure that makes it possible to examine interactively volume data, combined with a direct generation of geometric output. Example applications of *point-based exploration* include real-time isosurfacing and thresholding. The resulting visualization pipeline is purely point-based.

Universal representation and rendering - *Universal* point primitives are utilized for modeling surfaces, volumes and combinations of both. A general point-based rendering algorithm has been developed for handling universal point data and creating various 2D and 3D medical visualizations, including slice views, isosurfaces, MIPs and extensions. High performance is achieved due to a single-data-pass architecture and the utilization of modern features of programmable graphics hardware, resulting in the *deferred splat shading* technique. Normal vectors are estimated in image space using the GPU-based *micronormals averaging* algorithm.

Adaptive progressive refinement - A progressive refinement framework for point data has been developed. The refinement order and point density are controlled dynamically in a view-dependent fashion. The resolution of different portions of the virtual model can be adapted locally, e.g. with respect to the viewing distance or visibility. Smooth update is considered during interaction with the virtual model. The main focus is on medical scenes, consisting only of few complex objects and low depth complexity.

Sequential LOD data structure - A simple multiresolution representation has been developed for interactive rendering of large point models. The *sequential decomposition* approach requires only a reordering of the original point set without data duplication. The LOD selection is controlled by the aforementioned progressive refinement framework. No hierarchy information is introduced, providing direct access to the information in the data structure. All reordering algorithms are universal, including the *deflating spheres* algorithm and an approximate solution using binary space partitioning.

Outline

The work is organized as follows: In chapter 2 an overview of related work and state-of-the-art in point-based surface and volume visualization is provided, including multiresolution rendering. The point-based visualization pipeline and the universal representation and rendering architecture are presented in chapter 3. Moreover, in chapter 4 the point-based progressive refinement framework is described including the sequential level-of-detail solution. Application examples are presented in chapter 5, including a brief overview of the software architecture. Finally, chapter 6 summarizes the work, listing some ideas for future investigations.

Chapter 2

Related Work

In this chapter an overview of point-based representation and rendering solutions will be given, which are most relevant in the context of a typical medical visualization application. This includes interactive rendering of surfaces and volumes as well as techniques for preserving interactivity during interaction with complex models. For a detailed discussion of the advantages and limitations of point data, including comparisons to polygon-, voxel- and image-based representations, please refer to the available literature [BC03, Wan04, Gro06].

The first point-based approach for representing three-dimensional shapes in computer graphics has been the *particle system*. Since the beginning of the 1980s, it has been used to represent volumetric effects, such as smoke, fire and clouds [CHP+79, Ree83]. Each particle is visualized as a single point, associated with additional attributes like color, velocity and lifetime. In the following years this basic concept has been extended to procedural, implicit and physically-based modeling of complex objects [Ree85, WH94, DG95, BAKW03]. Examples are mass-spring models for soft tissue simulation and smoothed particle hydrodynamics for simulation of fluids and viscous material [DG96]. In recent years point data has evolved to a widely accepted modeling and rendering primitive in computer graphics. Points are used in many different areas of application, including data acquisition, sculpting and painting, simplification, animation, collision detection etc. [KB04].

2.1 Data Structures and Rendering Algorithms

In this section, a chronological overview of point-based visualization techniques will be presented, separated between surfaces and volumes. Although another popular approach for point-based rendering is raycasting, the main focus will be on *splatting* due to the direct relation to the rendering solution that is described in this work [AA03, WS05]. Important aspects will be the format of the point primitive, organization of points in a data structure, design of the rendering process and integration into an application-driven visualization pipeline. A crucial issue will also be efficiency, including interactivity and short preprocessing times.

2.1.1 Surfaces

The Evolution of Point-Based Techniques (1985-1997)

Modern point-based techniques were proposed by Levoy and Whitted in their technical report from 1985 [LW85]. Instead of using a rendering algorithm for each different geometry representation, they have suggested to utilize points as a *unique* rendering primitive. For the first time they have addressed some of the most important challenges when designing a point-based renderer, e.g. hole-filling and antialiased image reconstruction. Only more than 10 years later there has been a renaissance of the point primitive. The first reason has been the growing complexity of polygonal models, leading to sub-pixel triangle sizes. The second reason has been the advance in 3D laser range scanning technology, i.e. 3D digital photography. One example is the Stanford Digital Michelangelo project [LPR+00]. One of the first point-based approaches for surface visualization in the medical context is the *Dividing Cubes* algorithm, presented by Cline et al. in the year 1988 [CLL+88]. Their work can be considered as a direct extension of the *Marching Cubes* algorithm for isosurface extraction [LC87]. If a voxel cell is intersecting with the isosurface, it is projected onto the image plane. If its projection is at the size of a pixel or smaller, it is rendered as a point. Otherwise, the cell is subdivided as necessary.

Hole-Filling in Image-Space and Surfels (1998-2000)

In the late 1990s, the use of image space techniques, derived from image-based rendering, has become popular for closing the gaps between point samples during rendering. One example is the warping of a depth image to a new viewpoint during interaction with the scene. Shade has closed holes during the display of a warped image by assigning multiple depth points for each image sample [SGwHS98]. This representation, called *layered depth image* (LDI), can be considered also as a semi-structured point cloud. Another example for an image-based approach is the rendering architecture of Grossmann and Dally [GD98]. Their solution for the hole-filling problem during magnification and perspective projection is a hierarchy of depth buffers. The lowest resolution buffer has the same resolution as the target image. Each higher-order buffer has half the resolution of the lower one. During display, a z-buffer level is chosen, where the projected point spacing in pixel space matches its resolution. The final image is reconstructed by a hierarchical *push-pull algorithm*. In the pull phase a sequence of lower resolution color images is computed by averaging 2x2 pixel blocks. In the push phase the lower resolution images are used to compute the color for pixels, where a hole has been detected. Pfister et al. have extended the work of Grossman and Dally by introducing the *surfel* primitive, i.e. a surface sample, associated with depth, color, and normal attributes [PZvBG00]. Surfels are organized in an octree data structure, consisting of *layered depth cubes* (LDCs), which are extensions of the LDI concept. During rendering, the LDC octree is traversed until the target resolution is met. Then, the corresponding surfels are projected onto the screen. Visibility is resolved by the *visibility-splatting* technique. Texture filtering and interpolation of holes using a Gaussian kernel effects in an antialiased image reconstruction.

EWA Splatting and Hole-Filling via Dynamic Resampling (2001)

In 2001 point-based techniques were further developed in various directions in order to achieve a more flexible and efficient representation and improved visual quality. Based on the pioneering work of Levoy and Whitted, Zwicker et al. have introduced *surface splatting*, a point-based framework for high-quality antialiased rendering [ZPvBG01b]. One of their main contributions is a rigorous mathematical formulation of the *elliptical weighted average* (EWA) filter for anisotropic texture filtering. Their *deferred shading* renderer implements two passes. First, each point is projected as a radially symmetric Gaussian kernel to the screen space. The resulting ellipse is filtered and accumulated in screen space. In the second pass, all pixels are shaded. Additionally, surface splatting implements *order-independent transparency* by using a modified *A-buffer* algorithm for hidden surface removal and edge antialiasing. The design of Zwicker's primitive follows the common approach at that time, i.e. by storing normal and color information with each point. Kalaiah and Varshney have provided an extension of this concept. They have associated each sample with additional attributes from differential geometry, including the principal curvatures and directions [KV01]. Their *differential points* store more information per primitive, allowing shading with per-pixel accuracy and the modeling of a surface with fewer points by taking advantage of an efficient point-based simplification algorithm.

Another surface representation has been introduced by Linsen [Lin01]. The key idea of the so-called *point cloud* is the representation of a small part of the object surface by a point and its environment. This environment is defined by the nearest neighbor relation, combined with an angle criterion. The result can be visualized as a triangle fan using standard graphics hardware for rasterization. Several additional issues are addressed, like surface smoothing, up- and down-sampling, interactive multiresolution modeling and constructive solid geometry (CSG) operations. An alternative point-based modeling mechanism has been described by Alexa et al. [ABCO$^+$01]. Their *point set surfaces* are based on a local polynomial approximation of the underlying surface, using the *moving least squares* (MLS) approach. The object surface is defined by patches of polynomial height fields, each one given in a local reference frame. Alexa's solution to the hole-filling problem is a dynamic resampling procedure, i.e. by sampling the implicitly defined model until the desired screen space resolution is achieved. The result is a smooth visualization, even for viewing configurations with high magnification. Three years later, the work on dynamic sampling algorithms has been further developed by Guennebaud et al. [GBP04b]. They have presented a fast up-sampling method for high-quality rendering of point-based surfaces. Their main contribution is a new iterative refinement procedure that has been inspired by mesh-based subdivision surfaces. Points are organized in an octree and the refinement is executed on the included octree cells. Each step increases the number of points by a factor of four. Points are cached for further reuse, so this approach induces a significant memory overhead and after several refinement steps up-sampling cannot be performed interactively anymore. One year later, the same authors have improved the robustness of their refinement method and reduced the visual artifacts during rendering [GBP05].

Hardware-Acceleration and Point-Based Isosurfaces (2002-2003)

Most previously mentioned surface visualization techniques are purely software-based. Therefore, in the years 2002 and 2003 many researchers of the point-based community started to focus on performance optimization of the rendering process by taking advantage of modern graphics processor units (GPUs). The first example is Ren's et al. graphics-hardware-accelerated EWA surface splatting algorithm, including an object space reformulation of the screen space EWA filter [RPZ02]. Their multi-pass rendering procedure consists of visibility splatting, view-dependent EWA prefiltering and view-independent normalization (*2+1 rendering*). Object-space EWA surface splatting is using a variety of GPU-based functionality, including vertex and pixel shaders. Botsch and Kobbelt have further optimized the performance of hardware-accelerated EWA splatting by processing only one vertex per point during rendering using the *point sprite* OpenGL extension [BK03]. They have also reported a more efficient per-pixel normalization step by accumulating each splat contribution in an offscreen buffer. A similar technique was presented by Guennebaud and Paulin in 2003 [GP03]. Additionally, they have provided a GPU-oriented solution for per-fragment depth correction in order to improve the quality of the visibility splatting.

The aforementioned performance optimization techniques have had also an impact on a subarea of point-based surface rendering, which is of high importance for medical visualization. Bærentzen et al. has described for the first time a GPU-oriented point-based rendering framework dedicated to isosurfaces [BC03]. Points are extracted from volume data, based on a simplified version of the Marching Cubes algorithm. Normals are computed during the extraction using the central difference operator [Lev88]. Their graphics-hardware-accelerated renderer is based on the surface splatting technique and achieves high performance by using the fixed functionality pipeline of the GPU. Bærentzen et al. have compared their results to polygonal isosurface visualization and a software-based point renderer and have concluded that point-based isosurface rendering can be superior to previous techniques. Co et al. presented in 2003 a related approach, called point-based *iso-splatting* [CHJ03]. They have also used splatting for the visualization of their isosurface models. Additionally, their renderer is capable of projecting point samples onto the isosurface in order to improve visual quality by reducing the *thickening effect* of quadliteral splats. An approximate projection operation has been presented, which can be implemented on programmable graphics hardware. Beside positional and normal information, Co's point primitive requires the original scalar value at the location of each point sample.

Programmable Graphics Hardware and Deferred Shading (Since 2004)

In the years since 2004, many researchers have further refined point-based surface rendering techniques by taking advantage of the programmability of state-of-the-art graphics hardware. Zwicker et al. have reformulated their previous EWA surface splatting technique using homogenous coordinates in order to achieve perspective correct splat shapes [ZRB+04]. The replacement of the original affine approximation of the perspective projection avoids hole artifacts and improves image quality. Their point rendering

pipeline is implemented completely on the GPU using vertex and fragment programs. In the same year, Guennebaud et al. presented a high-quality point-based rendering technique for complex scenes by taking advantage of temporal coherency [GBP04a]. Their graphics-hardware-accelerated *point selection* algorithm limits complex shader computations only to visible points by extending the EWA multi-pass splatting approach. The framebuffer is used for the generation of an index vector that represents all currently visible points. An additional fast and simple rendering pass is required to generate those point indices. The index vector is used to speed-up the complex EWA shading pass and to accelerate visibility splatting by reusing the instance from the previous frame. Guennebaud's *deferred splatting* technique achieves an impressive maximum speed-up factor of 10 for complex outdoor scenes. Their implementation is slowed down by the data transfer between the GPU and CPU. The number of scene objects and points is limited by the 32 bit depth of the image buffer. Two years later the same authors have further developed their GPU-based rendering system, including perspective correct splatting, efficient approximation of EWA filtering, support for transparent point surface layers via depth peeling and smooth blending between point and polygonal primitives for hybrid visualization [GBP06].

In the year 2005, Botsch et al. described a surface rendering system that is using the newest feature of programmable graphics hardware at that time, including *multiple render targets*, floating point precision and advanced blending capabilities [BHZK05]. Although they have not introduced truly new concepts, their deferred shading renderer achieves high performance and provides per-pixel exact Phong shading by associating each splat with a linear normal map, extending the previous work in [BSK04]. The core routine consists of three rendering passes, including visibility splatting, the attribute pass and shading. The final lighting step is executed completely in screen space. The effective quality is comparable to the original purely software-based EWA splatting. Their point primitive consists of positional information, the surface's tangent axes and additional material properties. Kawata and Kanai have taken also advantage of state-of-the-art GPU-functionality and have introduced the *direct point rendering* technique [KK05]. Their point primitive contains only one attribute, a three-dimensional vector, describing the position of the corresponding sample. Such an approach can be considered as the purest point-based representation possible. The normal vector, required for lighting, is computed on-the-fly for each pixel from the depth buffer by fitting a plane through all neighboring depth points. All calculations are performed in a relative complex pixel shader, including the solution of a 3×3 linear equation system. Altogether, they require five rendering passes. Shading is performed in the final deferred shading step, as presented by Botsch et al. Kawata and Kanai have solved the hole-filling problem by scaling the size of the image buffer, based on the projected spacings between points. They have not addressed the problem of image quality degeneration, especially for low-resolution datasets and high visual magnification.

Finally, Zhang and Pajarola have developed a GPU-accelerated framework for point-based surface rendering that requires only a single geometry pass (*1+1 rendering*) [ZP06]. Their main idea is the partitioning of the point set into groups with minimal splat overlap. Within each point group, the splat blending can be omitted. In the first rendering pass, each group is visualized via simple splatting and the results are written to an associated

extended image buffer, including color, splat kernel weight and depth information. In the second pass, called *deferred blending*, all image buffers are composited. The grouping of the point set is performed during preprocessing and is reduced to a graph coloring problem. The authors have also described extensions for rendering of transparent models, mixing of opaque and non-opaque objects and simulation of multilayer refraction and reflection effects. Their solution is faster than the standard 2+1 approach but nevertheless achieves relative low point rendering rates.

2.1.2 Volumes

The Beginning of Point-Based Volume Rendering (1990-1995)

Most point-based volume rendering solutions are based on the volume splatting technique, which was introduced by Westover in the year 1990 [Wes90]. Like for most volume rendering techniques at that time, the input data structure is rectilinear and array-based. Nevertheless, with his work he has opened the way for the evolution from a discrete volume sample towards a modern point-based rendering primitive. Similar to the aforementioned surface splatting techniques, each sample point is associated with a low pass filter function for smooth image reconstruction. Westover has only considered orthographic projection, so all resulting screen space footprints can be precomputed in a table. The accumulated splat contributions are stored in several pixel buffers, called *sheets*. The final image is created by compositing all sheets buffer in back to front order.

Volume Splatting and the EWA Approach (1996-2001)

In 1996, Mueller et al. extended the basic splatting technique by introducing a ray-driven approach that allows perspective viewing [MY96]. Splats are intersected by rays that are sent from the image plane in object space. Such a technique makes it possible to optimize performance via early-ray termination. In the following three years the same authors have refined their volume splatter mainly by improving visual quality. The first example is an antialiasing technique for splatting [SMM+97], based on a scaling of the splat size with an increasing viewing distance. Each splat is rendered with basic hardware support, i.e. as an alpha-blended polygon. The second example is a method for the elimination of color *popping artifacts* during rotation of the rendered model [MC98]. Their solution is the utilization of image-plane aligned sheet buffers, resulting in a better approximation of the volume rendering integral. This is analogous to the incorporation of 3D texture mapping functionality for a texture-based volume renderer. In 1999, Mueller et al. presented a technique for reducing the blurry appearance of splatted object edges [MMC99]. Their approach is based on a reordering of the traditional splatting pipeline from a preshaded to a postshaded scheme. A performance optimization of the aforementioned volume splatter was presented by Huang and Mueller et al. one year later [HCSM00]. Their technique is purely software-based and is focused on fast rasterization by using compact look-up tables for the splat kernels. The authors have mentioned the implementation of their *FastSplat*

primitive on chip, but today their optimizations are probably less relevant due to the significant improvements in off-the-shelf graphics hardware since the year 2000.

A splatting technique, similar to Mueller's approach, was presented in 2001. Zwicker et al. have extended their EWA surface splatting framework to volume rendering [ZPvBG01a]. By incorporating the EWA volume resampling filter they avoid aliasing and blurring effects, especially for anisotropic volumes. For the first time they have discussed the utilization of a *universal* point primitive, i.e. a rendering primitive that can be used for both, surfaces and volumes. Zwicker's implementation is purely software-based and non-interactive.

Graphics Hardware Acceleration, VOTS and Uberflow (2002-2004)

In the following years, researchers have followed a similar way as for surface splatting and have focused on optimization techniques, provided by modern graphics hardware. In 2003, Nuber et al. presented a point-based volume visualization technique for large medical datasets [NBHJ03]. The heart of their system is an out-of-core data structure, which arranges point data in *color sets*. Each set is associated with a RGB color, enabling the possibility to explore the underlying volume data in real-time by selecting specific color sets as active. Their graphics-hardware-accelerated renderer takes advantages of the OpenGL point primitive, including positional and gradient information. The authors have not provided more details about their rendering procedure, including hole-filling, antialiasing and the problem of transparency sorting. Additionally, their system does not support interactive manipulations of the transfer function. In the same year, Xue and Crawfis proposed two splatting techniques for direct volume rendering, using OpenGL-based hardware acceleration [XC03]. The first technique is a vertex shader implementation of their original volume splatting renderer from 1993 [CM93]. Like most early hardware-accelerated splatting techniques, they use textured quads for rendering. For the incorporation of the low-albedo optical model, they require an explicit transparency sorting step. The second technique, coined *point convolution rendering*, is conceptually similar to EWA splatting. It is based on the X-ray optical model and uses standard OpenGL functionality. Although Xue and Crawfis need to traverse the voxel data only once, rendering is slowed down by the convolution operation, especially for large splats and high screen resolutions. The point convolution renderer does not support perspective viewing.

In the year 2004, Chen et al. presented a GPU-implementation of the original EWA volume splatter, utilizing programmable graphics hardware [CRZP04]. Their approach reveals some similarities to Xue's and Crawfis' work, including an axis-aligned splatting scheme and the utilization of textured quads. Their main contribution is the introduction of a distance-dependent filtering scheme for reducing the computational costs in case of minification and magnification. Additionally, they have presented a packing technique for their splat geometry based on the quantization of volume and footprint texture coordinates. The memory problem is not solved completely due to the overhead, resulting from the allocation of vertex indices. Therefore, Chen's et al. work targets only at rectilinear volumes with relative low resolutions (up to 256^3 voxels). Furthermore, the authors have mentioned a preclassification scheme for a culling of nearly transparent voxels. A GPU-

efficient solution for the elimination of color popping artifacts has not been presented.

Grimm et al. introduced in 2004 an alternative point-based primitive. Their *volume dots* (VOTs) are a functional representation of volumetric data [GBKG04b]. Each VOT is associated with coefficients of a Taylor series expansion for a given region of the dataset. The authors have taken advantage of the irregular sampling feature of a point-based representation by placing more and smaller primitives in regions of high interest. Although the paper is focused on volume modeling and data generation, Grimm et al. have also utilized the analytical processing capabilities of the VOT data structure for MIP volume visualization. They have reported that their approach is not well suited for complex datasets, which exhibit high variation among samples and if there are many regions with high importance. In the same year, Kipfer et al. revisited particle systems and presented a real-time particle engine that is implemented entirely on the graphics processor, including particle emission, motion, collision handling and rendering [KSW04]. *UberFlow* is also interesting for point-based volume rendering, because their visibility sorting technique is also realized completely in programmable graphics hardware. Their optimized Bitonic sort algorithm minimizes the number of instructions and texture operations, achieving rates of more than two million of particles per second. The size of their particle set is restricted by the maximally supported texture resolution.

Programmable Graphics Hardware (Since 2005)

Since 2005, many researchers have further developed volume splatting techniques by utilizing the programmability of modern graphics hardware. For example, Neophytou et al. have presented an optimized GPU-implementation of Mueller's image-plane aligned splatter [NM05]. Although, the sheet buffer data structure and therewith the visibility order is still managed completely in software, they provide solutions for the excessive overdraw and heavy vertex traffic problems that are common for most GPU-based splatting systems. Their performance optimizations focus on the fragment level, including early elimination of hidden splats and skipping of empty regions. Neophytou et al. have used various OpenGL features, including point sprites, optimizations for *early-z-rejection*, offscreen buffers and caching of point primitives in vertex arrays. Their system achieves interactive frame rates only for relative small volumes and point sets (up to one million of effective splats). One year later the authors extended their work by using ellipsoidal radial basis function kernels [NMM+06]. The utilization of this new rendering primitive can be considered as a refinement of the aforementioned radially symmetric approaches for volume splatting. It provides a more data aligned-fitting, resulting in a more efficient modeling of anisotropic volumes as well as scattered and irregularly distributed point datasets. The authors have used a 3D grid spatial data structure for an optimized localization of active splats that intersect with the current sheet buffer slice and facilitated front-to-back data traversal for composited rendering. Due to the complex shape of their primitive they have to store additional attributes per point, resulting in a memory overhead. Neophytou et al. have again reported interactivity only for relative small point sets with up to 200 thousand of primitives.

2.2 Techniques for Preserving Interactivity

One of the main challenges for the visualization of large medical datasets is the preservation of interactivity. One popular solution is the reduction of the detail level during interaction in order to guarantee fast rendering rates. In this section a chronological overview of the most relevant *output sensitive* rendering techniques will be given, of which the time complexity depends only weakly on the complexity of the model.

The missing connectivity information of point data and the resulting flexibility open new perspectives for the development of new multiresolution techniques, especially for an interactive environment. Although there are many polygon-, volume- and image-based solutions for that problem, most of these techniques imply a loss of the flexibility feature [Wan04]. Typical drawbacks are also the non-trivial integration into a graphics-hardware-oriented setting, long preprocessing times and lacking of support for the interactive manipulation of the data. Therefore, the focus in this section will be on point-based multiresolution approaches. These can be separated into techniques for surfaces and volumetric datasets, whereas most researchers from the point-based community have focused on level-of-detail (LOD) techniques for surfaces. For a general overview, please refer to the surveys of Sainz and De Floriani [SPL04, SP04, FKP04].

2.2.1 Surfaces

Large Surface Models and Splats - QSplat (2000)

In the year 2000, Rusinkiewicz and Levoy presented *QSplat*, a point-based rendering system for 3D-scanned surface models [RL00]. Although QSplat is one of the first approaches that take advantage of graphics-hardware-accelerated point rendering, the authors' main focus has been on multiresolution visualization of very large datasets. All points are stored during preprocessing in a bounding sphere tree. Inner nodes contain average values for position, normal and color. During rendering, their algorithm traverses the hierarchy until the current bounding sphere radius is below a predefined splat size. The resulting splats with constant size are drawn in immediate mode, using OpenGL functionality. Additionally, the authors encode all point primitives to 48 bits, using an efficient quantization scheme. One year later Rusinkiewicz et al. extended their work to *Streaming QSplat* for view-dependent streaming transmission of the point hierarchy over a network [RL01].

Dynamic Sampling Techniques (2001)

In the following two years many researchers have concentrated on multiresolution techniques that are based on dynamic point sampling. A first example is the work of Stamminger et al., a sampling method for procedural and complex surfaces [SD01]. Their $\sqrt{5}$ hierarchical sampling scheme adapts the point density locally, according to the currently projected splat size of a procedurally defined surface area. For each initial point, four new neighboring points are generated recursively as long as undersampling is detected. Their sampling scheme guarantees a quite even distribution of the surface points. For complex

objects, they have proposed to precompute a sequential list of quasi random samples on the surface and to use only its prefix for level-of-detail rendering. At each frame, they require the calculation of the prefix length, based on the object surface area and its camera distance. Choosing a large enough length results in a hole-free image reconstruction. A very similar multiresolution approach has been implemented also by Deussen et al. for the point-based visualization of complex plant models [DCSD02]. Another example for the dynamic sampling approach has been presented by Wand et al. [WFP+01]. The *randomized z-buffer* algorithm requires a triangle-based scene as input. Random sampling is used to generate the surface points for visualization. The corresponding point density is computed for each triangle, based on the *projection factor*, i.e. an estimation of the projected area. The calculation of the target number of points is based on a probabilistic urn model. The authors have presented various optimization techniques, including octrees for spatial acceleration, distribution lists for efficient random selection of triangles and a caching mechanism for generated sample points. Their system is very memory consuming and results have only been presented for scenes with replicated object instances.

Compact Representation, Hardware-Orientation and Simplification (2002)

In the year 2002, Botsch et al. presented a memory efficient hierarchical representation for point-based surfaces [BWK02]. By using uniform clustering and quantization techniques, point positions can be encoded with less than two bits. Their octree data structure is used for visibility calculations and backface culling via normal cones. Breadth-first traversal allows the reconstruction of the surface model on a lower refinement level. Moreover, the hierarchical organization of the point data makes it possible to implement fast incremental 3D transformations. Botsch et al. have described an efficient rendering procedure, based on splatting. Their system is purely software-based and a straight-forward realization with hardware support is not possible. On a similar subject, Coconu and Hege have presented a point-based surface visualization method that combines both, a level-of-detail representation and graphics-hardware-oriented rendering [CH02]. Their LOD algorithm is similar to Pfister's surfels technique, utilizing also a hierarchical representation with points that are stored in three orthogonal layered depth images at different resolutions. Their hybrid octree-based data structure contains points and triangles, similar to Chen's point and polygon rendering system *POP* [CN01]. Their renderer switches to triangle rendering, whenever the estimated point density gets too low. Pauly et al. presented different point surface simplification techniques [PGK02]. Simplification is typically performed during preprocessing for a one-time reduction of the model complexity. It is mentioned here due to its close relation to point-based multiresolution rendering. Some researchers have referred to Pauly's et al. work for the construction of their level-of-detail data structure. Three types of algorithms have been presented, including *clustering* (replacement of a point cluster by one representative sample), *iterative simplification* (collapsing of point pairs) and *particle simulation* (distribution of points, based on inter-particle repelling forces). The authors have used local variation estimation for concentrating more points in regions of high surface curvature. All methods resample the input surface, i.e. create a new point set.

Data-Driven Hierarchy, Sequentialization and Statistical Analysis (2003)

A progressive representation of point-based surfaces has been presented by Fleishman et al. [FCOAS03]. Based on Alexa's work from 2001, the authors have constructed a multiresolution model by using a projection operation, which maps points in the proximity of the surface onto a local polynomial approximation. Each level is constructed iteratively from the previous one by inserting new points. Fleishman et al. have concentrated on memory-efficient progressive encoding of point sets. They have not provided details about the integration of their approach into a point-based rendering pipeline. In the year 2003, Pajarola focused on fast computation of point hierarchies [Paj03]. Instead of utilizing the widely used space-driven region-octree, the author has incorporated an adaptive data-driven octree, resulting in fewer tree nodes. The recursive top-down generation of the data structure includes the computation of the spatial partitioning of point locations, average per-node attributes (normals and colors) and splat sizes. Each point primitive requires 46 bytes. Similar to the aforementioned surface splatting techniques, a high-quality multi-pass renderer is used based on programmable graphics hardware and alpha-textured polygons for the smooth blending of splatting kernels [PSG03]. Pajarola's et al. system was published also under the name *Confetti*, one year later [PSG04].

The first paper on graphics-hardware-accelerated sequential multiresolution rendering was published by Dachsbacher et al. in the year 2003 [DVS03]. The *sequential point trees* data structure is implemented completely on the GPU by taking advantage of its programmability. A QSplat-like hierarchy is linearized and the rearranged nodes are stored in a sequential array. The array is sorted by the maximum viewing distance and each node is associated with the corresponding minimum value. Only a prefix of the array needs to be processed by the graphics card. The length of the prefix is computed on the CPU and controlled by a view-dependent geometric approximation error. In contrast to previous hierarchical approaches, culling of complete subtrees is not possible, but the full exploitation of hardware-acceleration results in very high splat rates and the CPU is free for non-rendering purposes. Nevertheless, a performance overhead can be observed, because up to 40% of the points need to be culled by the GPU. Additionally, the input is still an octree-based data structure that requires the generation of average point information. Dachsbacher's et al. data structure has also been utilized for Pajarola's et al. out-of-core visualization system for large point-based surfaces, called *XSplat* [PSL05].

An alternative approach is Kalaiah's and Varshney's method for the modeling of point-based surface geometry using statistical analysis [KV03]. Their main idea is the creation of an octree that does not contain any explicit point attributes. Instead, each node is associated with statistical parameters, resulting from a *principal component analysis* (PCA) of the input model. At rendering time the octree is traversed in a view-dependent manner and the PCA parameters, such as the orientation frame, mean value and variance, are used to generate point data with a Gaussian number generator. Kalaiah's and Varshney's approach handles position, normal and color information and exhibits a compact representation with only 12 bits per octree node. Nevertheless, it requires hours of preprocessing time and the rendering performance degenerates significantly for close-up views.

FastFPS, Layered Point Clouds and Progressive Splatting (Since 2004)

Moenning and Dodgson have presented a simplification algorithm, called *FastFPS*, that is suitable for a progressive representation of point-based surfaces [MD03]. Their work is based on the *Farthest Point Sampling* strategy by Eldar et al. for irregular and progressive image sampling [ELPZ97]. Their main idea is the placement of the next sample point "in the middle of the least-known area" in the sample domain. It is implemented by an incremental bounded Voronoi diagram construction. Moenning's and Dodgson's extension to sub-sampling of point surfaces is based on the computationally and memory efficient *Fast Marching* algorithm. It exhibits a uniform point distribution with user-controlled density guarantee. The integration into a point-based rendering pipeline has not been discussed. FastFPS has been further developed to MLS-based point cloud resampling [MD04, Moe06].

In the year 2004, Gobetti and Marton developed a view-dependent multiresolution structure for very large point surfaces (e.g. 234 mio. of samples) [GM04]. Their work is based on a hierarchy of object-space point clusters, called *layered point clouds*. Each cluster consists of thousands of compressed samples, amortizing the level-of-detail extractions costs. In contrast to most previous techniques, the final multiresolution model is composed of exactly the same points as the input model. The core data structure is a binary tree, created in a top-down fashion. For each node a subset of points is constructed via uniform sub-sampling, approximated by random picking of points. The remaining points are processed for each subtree by a simple bisection operation along the longest bounding box-axis. Additionally, each node is associated with an average sample spacing value for hole-filling as well as a bounding sphere and normal cone for view-dependent LOD control and backface culling. During rendering, the tree is traversed in breadth-first order and the included point clouds are accumulated until the required density is reached. Progressive refinement is used to hide data access latency. The authors have exploited frame-to-frame coherence for occlusion culling by using the rendered points from the previous frame as the occluder for the current frame. The simple point cloud construction mechanism may lead to visual artifacts. Moreover, efficient modification of the point data (e.g. for surgical cutting) is not possible due to the hierarchical nature of the approach. The layered point clouds method requires hours of preprocessing time for larger datasets.

In the same year, Wu and Kobbelt presented a technique for the conversion of a dense point cloud into a sparse set of splats [WK04]. Their decimation procedure considers the full circular or elliptical splat geometry. It is based on a global relaxation procedure, which replaces iteratively splat subsets with smaller and better distributed sets. Referring to Pauly's work from 2002, such an approach reveals many similarities to particle simulation. The long processing times of the algorithm are compensated by their high quality decimation results. In the year 2005, the same authors have presented the *progressive splatting* technique. The main idea is the representation of the multiresolution model as a base splat set and a continuous and ordered list of *detail operators*, similar to Hoppe's progressive meshes [Hop96, WZK05]. Wu et al. have utilized a more efficient greedy approach for decimation related to Pauly's iterative simplification. Both works focus only on the data-processing part of the visualization pipeline instead on interactive rendering.

2.2.2 Volumes

Hierarchical Splatting (1991-1999)

One of the first splat-based multiresolution techniques for volume rendering is the *hierarchical splatting* algorithm by Laur and Hanrahan, introduced in the year 1991 [LH91]. Their core data structure is a pyramidal representation of the volume, implemented with an octree. Octree nodes at different levels are drawn using splats. Lower resolution levels are associated with fewer and larger primitives. For such cases, the authors have provided a solution for opacity correction during composite rendering of transparent splats. Laur's and Hanrahan's approach supports *progressive refinement* for preserving interactivity, controlled by user-supplied error criteria.

Previewing Large Volumes (2000-2002)

In the year 2000, Kreylos et al. presented a multiresolution previewer for large arbitrary volumes meshes [KMH00]. The original dataset is approximated by a point cloud. Their main idea is the iterative application of a point decimation operation, based on the *maximum independent sets* algorithm. The algorithm requires a Delaunay tetrahedrization of the input point set. The result is a chain of point subsets, which can be utilized for progressive transmission. Their multiresolution representation requires no explicit hierarchy and each point is stored exactly once without additional information. For rendering, Kreylos et al. have taken advantage of an image-aligned sheet buffer data structure. Instead of splatting, all sheet points are triangulated and rendered as OpenGL polygons, resulting in relative low frame rates. Detailed results for their multiresolution technique, like the granularity of the resolution levels, have not been presented.

Two years later, Wilson et al. presented a visualization system for large-scale particle sets [WMQR02]. Their main contribution is an octree data structure and rendering technique for a hybrid representation of the input volume. Data areas with high particle density are displayed with a texture-based volume renderer. Low-density areas are visualized using simple point-based rendering. Such an approach has two advantages: first, a fast visualization of compact data portions, and second, a fine visual resolution for detailed structures. Their mixed point- and volume-based rendering method refers only to particle beams for accelerator design, but represents an interesting alternative for hierarchical volume visualization.

Cluster Hierarchies, Ellipsoidal Decomposition and Monte Carlo (Since 2003)

In the year 2003, Hopf and Ertl proposed a hierarchical splatting technique for scattered point data without surface information [HE03]. Although the background for their work has been the visualization of particle systems and astronomical datasets, it has some characteristics of a point-based multiresolution volume representation. Their main idea is the creation of a hierarchy of point clusters, based on a PCA clustering procedure. Such a data-driven approach has better spatial adaptation properties compared to traditional

octree-based space subdivision. Each point cluster is represented as a sequential array for efficient processing and graphics-hardware-accelerated rendering. Transparency sorting is only supported in software. Hopf and Ertl have reduced the memory requirements by utilizing quantization of relative point coordinates. Each point primitive is associated with fixed color and size values, so transfer functions are not supported. The point size is precalculated, based on the astronomical radiant flux. The authors have not considered the hole-filling problem for small-scale objects, as required for medical visualization. A similar approach is Co's et al. hierarchical clustering procedure for scattered volume data [CHH+03]. Their multiresolution approach is also based on the PCA for the generation of the *cluster binary tree* for point data. Space subdivision operations are scheduled in a priority queue, based on a maximum scalar deviation error metric. Each node in the hierarchy is represented by a center point for the current cluster. Its scalar value is evaluated using a scalar function approximation, derived from a composition of radial basis functions. The level-of-detail extraction is based on a depth traversal of the tree, optionally with error control. Information about the rendering technique has not been provided.

Another data-driven multiresolution representation for volumes has been introduced by Welsh and Mueller [WM03]. Their main contribution is a feature-sensitive decomposition of the data into a hierarchy of ellipsoidal point primitives. The construction of their tree data structure is based on a frequency analysis of the input volume. The adaptive distribution of the point samples preserves the original frequency characteristics and allows a representation of the original object with a significantly reduced number of points at adequate quality. Uniform areas are represented with fewer and larger points. Welsh and Mueller have pointed out that time-critical algorithms can impose problems for multiresolution rendering due to variations of the tree traversal depth between consecutive rendering frames. Therefore, they have considered also solutions for smooth update during interaction in the context of view-dependent level-of-detail. Nevertheless, their system achieves quite low frame rates and the preprocessing requires hours of time, even for small datasets with only 256^3 volume samples.

A point-based X-ray volume rendering method, based on *Monte Carlo integration*, has been presented by Csébfalvi [CSK03] et al. Their idea is the extraction of the most relevant information from the volume data using random-based importance sampling. A point cloud is generated by choosing random samples with a probability that is proportional to the voxels' density values. During rendering, the accumulated density contributions at each pixel value are normalized with respect to the total point count and used as an estimation for the final X-ray intensity values. The precomputed point cloud can be utilized directly for progressive refinement by rendering a smaller subset. It has been shown that time and memory complexity depends only on the number of pixels in the image plane. The renderer implementation is only software-based due to the limited floating-point precision of graphics hardware. Transfer function control for *Monte Carlo Volume Rendering* was integrated one year later [Csé04]. Update operations require several seconds and are therefore not appropriate for medical applications.

2.3 Discussion

In the following, the major features of state-of-the-art point-based visualization and multiresolution techniques will be summarized. This chapter will be concluded by presenting some limitations and by underlining the aspects that differ most from the system described in this work.

2.3.1 State-of-the-Art

Splatting can be considered as the most popular visualization technique for point-based data. It makes it possible to achieve a well balanced trade-off between rendering speed and visual quality. Elliptical splat shapes and Gaussian reconstruction kernels play an important role for high-quality display due to their good approximation properties for surfaces and anisotropic volumes. The hole-filling problem is solved by scaling the projected point size in order to close empty regions in the point cloud. Another solution that has won much recognition in the last years is the local increase of the point density by utilizing appropriate upsampling algorithms.

The utilization of graphics hardware has gained much importance in interactive computer graphics. The point-based community has also followed this trend by enhancing the point-based rendering pipeline using programmable GPUs. Recent implementations include graphics-hardware-accelerated occlusion culling, anti-aliasing, deferred shading, per-pixel lighting computations etc. Late research is focused also on the full integration of the processing and rendering pipeline on the graphics processor, including inter-particle interaction and transparency sorting.

Most point-based rendering systems are based on a hierarchical data structure, like the octree and binary space partitioning tree. The hierarchical nature of such approaches effects in many advantages for multiresolution modeling, visibility calculations via normal cones and sub-tree culling as well as handling of large datasets. To amortize the per-point overhead and to take full advantage of the stream-based architecture of the graphics processor, some researchers have started to consider sequentialization of the point data structures and clustering of point elements in linear point arrays.

Finally, state-of-the-art multiresolution techniques have began to go off the beaten track of regular space partitioning. Current techniques incorporate data-driven and feature-sensitive models for an adaptive construction of the level-of-detail representation. Alternative and non-deterministic approaches have gained also more popularity, implementing e.g. importance-based sampling of the point data and the utilization of statistical tools.

2.3.2 Limitations

Only a few point-based rendering solutions have been applied in the medical context. One reason is surely the domination of traditional visualization approaches, like triangle-based rendering, volume raycasting and texture-mapping-based techniques. Another reason is the origin and the motivation of point-based methods, i.e. particle systems and

three-dimensional photography. Typically, integration into an application-driven medical workflow is not the main aspect of the aforementioned related work. Possible limitations are the lack of flexible point data structures that can be used in various stages of the data-processing pipeline, restricted configurability of the renderer component, non-modular software design etc. Moreover, several aforementioned techniques focus only on a specific aspect of the visualization pipeline, neglecting important functionality, like the handling of multiple objects in the scene, incorporation into the application's interaction system, support of color and transparency changes and stable memory allocation.

Although related work has shown that point-based surface visualization can be an attractive alternative to triangle-based rendering, there are some limitations. Many techniques focus on high-quality visual output at the expense of interactivity. This is evident especially for the early software-based solutions. Moreover, high-quality approaches require often additional per-point information, e.g. the tangential axes for elliptical splat primitives. Often, the resulting memory overhead cannot be justified, in particular for noisy medical datasets. Another aspect is the typical structure of today's point-based surface splatters. The multi-pass approaches require the processing of the point dataset multiple times per frame. Such a strategy complicates the incorporation of efficient progressive refinement control. This problem can get even worse for large datasets in an out-of-core setting. Additionally, almost all surface representations require the association of each point primitive with a normal vector. Therefore, points loose their useful symmetry property, resulting in a significant memory overhead and complication of many algorithms, like simplification and data modification.

Many of the aforementioned volume splatting solutions are not pure point-based techniques. The input data structure is still a regular and grid-based volume. These approaches can be utilized very well for high-quality volume rendering, but they cannot benefit from the point-based modeling features, like adaptive sampling and simple multiresolution control. Moreover, some techniques require additional memory intensive data structures, e.g. a gradient volume. Similar to early point-based surface visualization approaches, many implementations are purely software-based and interactivity can often only be guaranteed for smaller-sized volumetric datasets.

When considering the related work on point-based level-of-detail, it can be observed that the majority of researchers have concentrated on surface representations. Only a few solutions could be utilized for handling both representations, surfaces and volumes. Another important aspect is the domination of hierarchical data structures, making sequentialization a hard task. Usually, average information has to be computed for each node in the hierarchy. This imposes new challenges with respect to GPU-oriented implementation and interactive manipulation of the included point cloud. Some researchers have tried to solve this problem by caching the extracted points in linear data structures. Typically, this approach requires additional update operations for each rendering frame, inducing another performance overhead.

Objectives

An important goal of this work has been the design of a component-based software library that can be utilized for fast development of medical visualization applications. Therefore, a point-based visualization system has been developed with the following functionality:

- Pure point-based representation with minimal number of attributes per point
- Non-hierarchical data structures and sequentialization of point data
- Direct access to point data without resampling and average information
- Configurable single-pass renderer that handles both, surfaces and volumes
- Utilization of programmable graphics hardware and deferred shading
- Progressive refinement control, customized for objects in a medical scene
- Simple and efficient multiresolution approach for sequential point data
- Component-based software design for maximal reusability and extendibility

Chapter 3

Point-Based Visualization Pipeline

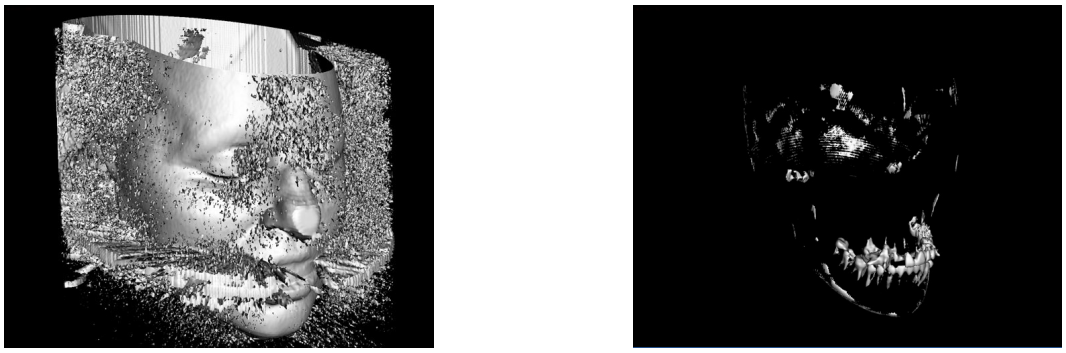
In this chapter the major building blocks of a point-based visualization pipeline for medical applications will be described. This includes interactive exploration of volume data and an efficient and general graphics-hardware-accelerated rendering solution for surface-based and volumetric datasets. Finally, results will be presented for data-processing and visualization components.

Visualization plays a fundamental role for the modern medical workflow and it is utilized at different stages of the processing pipeline. Consequently, the focus will be on memory-efficient geometric data structures, point generation algorithms and interactive rendering techniques. Advanced preprocessing techniques, such as filtering and segmentation as well as further pipeline stages, like navigation and simulation, are not considered in this chapter. Furthermore, the concentration will be on medical applications that require a high degree of interactivity during exploration of the virtual patient model, e.g. for diagnosis and surgical planning. The key points of the point-based processing pipeline in this work are as follows:

1. **Import** - The first step includes the import of medical volume data, typically in the DICOM or raw format [Rev97]. The resulting voxel data is stored in a traditional volume data structure with support for external memory storage. This is an important requirement in the context of large datasets, i.e. if the data size exceeds the amount of available main memory.

2. **Generation** - In the second step a point-based *exploration data structure* is constructed, based on the scalar values from the input volume. It can be interpreted as a dynamic 3D model and utilized directly for rendering and visual inspection of the underlying information.

   The exploration data structure is generated for a user-defined *exploration range*. A large range implies more points that are available for interactive examination and visualization. In practice it may be hard to estimate a reasonable range a priori and therewith achieve an optimal trade-off between the memory consumption and the flexibility of the exploration. For such cases, it is suggested to split the generation into two sub-steps:



(a) Too low isovalue (b) Too high isovalue

Figure 3.1: Isosurface raycasting of a human head with low information content.

(a) *A priori generation* - This sub-step includes the computation of the exploration data structure for fast previewing. This configuration comprises a large exploration range and a coarse model resolution level. The processing time should not exceed a few seconds in order to prevent significant delays in the medical workflow.

(b) *A posteriori generation* - The generation of the exploration data structure is re-executed at high resolution and a reduced exploration range, available from the previous sub-step. From a technical point-of-view the same processing components are utilized, but with a different input configuration.

A priori generation can be considered as a manual preselection step for the exploration range and a posteriori generation as fine-tuning. Memory reduction can be achieved in both sub-steps. In 2a, the number of points is trimmed down by computing a low quality representation, e.g. via sub-sampling of the volume. In 2b, advantage is taken of the irregularity feature of point data and the memory consumption is minimized by using a smaller exploration range [NBHJ03].

Using a full exploration range during a priori generation may unnecessarily complicate the generation procedure, because usually only specific subranges contain useful information. It makes sense to choose a small as possible range, discarding data portions that are not relevant for the current application (figure 3.1). (Semi-)Automatic preselection algorithms could be a useful tool to support the user in such situations. One example is the utilization of *Hounsfield units* by mapping specific scalar ranges to anatomical structures, like skin, soft tissue and bone [Hou73]. Another example is the exploitation of data-driven approaches from semi-automatic transfer function design by locating boundaries of interest in the data value space [KD98].

3. **Fixation** - The exploration is controlled by an *exploration parameter* that is defined with respect to the exploration range. During the whole inspection process it can be

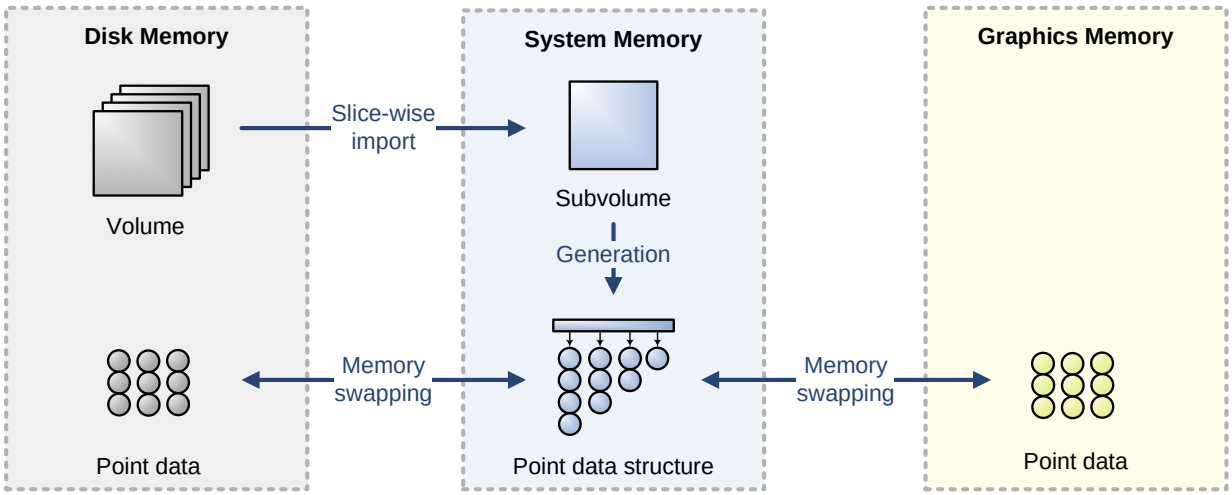


Figure 3.2: Point-based data-processing on different levels of memory.

modified interactively, effecting in a variety of 3D point-based visualizations. Afterwards, the user has the possibility to *fix* the exploration data structure, i.e. extract all relevant point elements into an *operation data structure* and free the data that is no longer needed. This can be considered as an optimization stage. Further exploration, as mentioned in step 2, is therewith not possible anymore. Then, the final point-based model can be used for additional application-specific steps of the medical processing pipeline.

One crucial aspect in medical processing is the efficient and stable handling of large datasets. Therefore, all algorithmic components of the presented visualization pipeline can be executed at different levels of memory, including graphics, system and disk storage (Figure 3.2). Moreover, after a posteriori generation of the exploration data structure, steps 2 and 3 do not require the access to the original volume data, because all relevant information can be represented in point-based fashion. The original voxel data could be discarded from the pipeline. Although such a strategy is not typical for medical applications, it is an attractive option for freeing extra memory.

Notation

In the following, the notation and technical terms are introduced that are used in this chapter. A volume is considered as a structured grid of *samples* in three dimensional space. Each sample is associated with a scalar intensity value $x \in \Re$ in the range $[r_{min}, r_{max}]$. Applying a reconstruction function, e.g. trilinear interpolation, creates a continuous *scalar field*. Additionally, neighboring samples specify a subvolume, called *cell*. For example, a cubic cell that is aligned to the volume grid is constructed by eight corner samples. The *cell range* $[x_{min}, x_{max}]$ is defined using the minimum and maximum scalar values. The *span* is the width of the corresponding interval, i.e. $s = x_{max} - x_{min}$.

3.1 Exploration of Volumes

The term "exploration" can be interpreted in various ways. One example is diagnostic or surgical exploration, describing a pure medical procedure. Another example, as mentioned in chapter 1, is the exploration of the virtual patient model. It is a crucial part for the understanding of 3D volume data and the underlying spatial scalar field, targeting at the examination of characteristic structures inside solid bodies, like bone or soft tissue. Virtual exploration refers to a step in the medical visualization pipeline and comprises the utilization of a variety of computer graphics tools, including:

- Interaction with the 3D model by changing the viewing position and angles
- Adjustment of color or transparency settings using a transfer function
- Revealing interior structures by cutting away parts of the model
- Measurements of distances, angles and volumes in 3D space

Many applications require an explicit geometric representation of the explored model. Therefore, *point-based exploration* is considered in this work as a dynamic point set generation procedure, controlled by a variable exploration parameter. In the following a more formal definition will be provided, including examples and a description of the corresponding exploration data structure.

Exploration Parameter Set

Let $\mathfrak{X}$ be the set of possible exploration parameters. It is crucial to define the parameter set with respect to its practical realization: First, it should be possible to control the exploration parameter intuitively in the graphical user interface (GUI) of the application. Second, it should allow the implementation of an efficient exploration data structure. Therefore, a closed *interval-based* parameter structure has been chosen, i.e.:

$$\mathfrak{X} = \{\ [x_1, x_2] \subset \Re \mid x_1 \leq x_2\ \} \tag{3.1}$$

Such a 2D parameter structure makes it possible to implement a variety of exploration functions for the medical visualization pipeline. Additionally, both parameter values x_1 and x_2 can be specified easily by non-technical experts e.g. by two slider controls in the application's GUI. The currently selected exploration parameter is called *target* parameter $\Theta = [\theta_{min}, \theta_{max}] \in \mathfrak{X}$.

Activation Function

Let P_{IN} be the set of input points, generated from a volume dataset. The activation function $A : P_{IN} \rightarrow \mathfrak{X}$ associates each input point with an exploration parameter. During exploration, a point $p \in P_{IN}$ is considered as *active* for the parameter $X \in \mathfrak{X}$, if the following condition is true:

$$A(p) \cap X \neq \emptyset \tag{3.2}$$

Only active points contribute to the visualized point model. For the interval-based parameter structure, $A(p)$ has the form $[\alpha_{min}, \alpha_{max}]$. The values α_{min} and α_{max} are called *activation values* for point p. The final definition of the activation function is application-dependent. Some examples will be provided in the remainder of this section.

Exploration Function

Let $\mathcal{P}(P_{IN})$ be the power set of P_{IN}. The dynamic generation of the point set during the exploration procedure is described by the point-based *exploration function* $E_A : \mathcal{X} \rightarrow \mathcal{P}(P_{IN})$. It is defined by the following relation:

$$E_A(X) = \{\ p \in P_{IN} \mid p \textit{ is active for } X \textit{ with respect to } A\ \} \tag{3.3}$$

$E_A(X)$ is called the *active point set* for the parameter X with regard to the activation function A. E_A is implemented by the exploration data structure, as will be described in sub-section 3.1.1.

Examples

The exploration framework and the original development of the exploration data structure have a close relation to the discipline of accelerated *isosurface* extraction [vRLHJ+04]. An isosurface is a surface-based representation, which consists of a set of geometric primitives that are associated with a constant scalar value. Many solutions are based on a one-time generation of the surface primitives, i.e. for each new isovalue previous geometry is discarded. Such a strategy requires a complete recalculation of the virtual 3D model, usually combined with a costly reinitialization of the associated renderers. Moreover, standard triangle-based methods, like the Marching Cubes algorithm, create a set of interconnected triangles [LC87]. This may lead to high memory consumption, making isosurface exploration and memory management a hard challenge.

Many researchers have concentrated on the improvement of the isosurface extraction process. Acceleration is mainly achieved by reducing the number of volume cells visited during the extraction or by providing an efficient access to active cells. There are three basic types of techniques: domain search, range search and cell propagation [WvG92, CMM+97, BPS96]. Advanced techniques include also incremental exploration, view-dependent extraction and GPU-based implementation [CVCK03, LT04, KW05]. Nevertheless, achieving interactive exploration of large volume datasets with explicitly generated isosurface geometry remains a major challenge.

To overcome performance and memory drawbacks of traditional isosurface generation techniques, interactive isosurfacing (sometimes called *interactive interrogation*) is interpreted in this work as a point-based exploration process. Assume that each volume cell is approximated by a point $p \in \Re^3$, located at the cell center, and x_{min}, x_{max} are the scalar

(a) Surface (b) Extended surface

Figure 3.3: Visualization of a mandible without (a) and with (b) volumetric information.

values corresponding to the cell range. Then, the activation function for isosurface exploration is simply $A_{iso}(p) = [x_{min}, x_{max}]$. Based on equation 3.2, the activation of the point p corresponds to the intersection of the associated cell with the isosurface, i.e. for an isovalue υ there is $[\upsilon, \upsilon] \cap [x_{min}, x_{max}] \neq \emptyset \Leftrightarrow \upsilon \in [x_{min}, x_{max}]$.

The interval-based nature of the exploration parameter space makes it possible to derive a direct extension of isosurface exploration. By using an *isovalue range* $[\upsilon_{min}, \upsilon_{max}]$ with $\upsilon_{min} < \upsilon_{max}$ for point-based exploration, it is possible to extract *extended surfaces*, i.e. "thick" surfaces including volumetric information (figure 3.3). The resulting application can be interpreted as a kind of *interactive thresholding* [RBH+04]. Other application examples include keyframe-like animation of point-based models as well as front propagation algorithms (e.g. *interactive region growing*) by defining the activation function based on the keyframe and process time, respectively. Notice that such applications may require the duplication of point instances, if points are associated with multiple activation intervals. Point-based isosurface exploration will be used as the standard example for the remainder of this section.

3.1.1 Data Structures

In this section the exploration and operation data structures will be described. Their design is inspired by the requirements and objectives, formulated in the introduction and related work chapters, respectively. Hence, the principles of point-based data structure design for interactive medical visualization applications are defined in this work as follows:

Universal - One data structure should be used for both, volume and surface models, in order to avoid heterogeneous component design, development of multiple processing and rendering components as well as conversion between different data formats [LW85].

Sequential - Point data should be encapsulated in maximally large continuous chunks of memory. The resulting point arrays facilitate cache-friendly GPU-oriented pro-

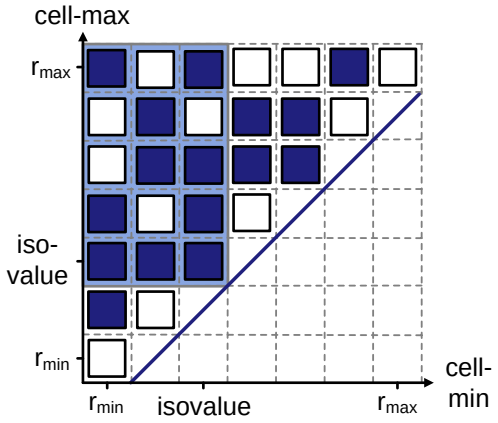


Figure 3.4: Discretized span-space. The small squares represent empty (white) and non-empty (blue) volume cells. The light blue area illustrates active cells for the current isovalue.

cessing. Other researchers call such data structures also *linear*, *plain* or *stream-based* [GPZ+04, GM04, Paj05].

Direct - Fast processing and interactive manipulation procedures require immediate access to the point data. The suggested solution is the abandonment of hierarchical and incremental encoding schemes as well as averaging and duplication of information.

Exploration Data Structure - Span-Triangle

The design of the exploration data structure, called *span-triangle*, is based on a special scalar range representation that has been designed for accelerated isosurface extraction [vRLK04]. A volume cell with a scalar minimum x_{min} and maximum x_{max} is considered as a point (x_{min}, x_{max}) in $\Re^2$. The corresponding two-dimensional space with cell-minima on the x-axis and cell-maxima on the y-axis is the so-called *span-space* [LSJ96]. In contrast to the cell interval representation $[x_{min}, x_{max}]$, it is possible to subdivide the data domain in a simpler way and to develop efficient search algorithms for active cells [SHLJ96]. In this work, a discretized version of the span-space is considered due to the focus on medical volume datasets (e.g. CT, MRI), which typically contain only discrete scalar values (figure 3.4).

The development of the data structure has been inspired by the *span filtering* and *ISSUE* algorithms [Gal91, SHLJ96]. Their representations are also based on a regular decomposition of the span-space, whereby in this work a different subdivision scheme and a different data structure for storing relevant information is used. Additionally, once the data is stored in the span-triangle, it does not require any explicit searching for active points. It establishes a tight coupling between representation and point-based rendering, which makes it possible to exploit data coherence between different explored models and to take advantage of incremental update algorithms. This results in the smooth manipulation of

the exploration parameter and interactive rendering rates. A direct extension of the data structure to floating point scalar values, as required e.g. for data from computational simulations, has been presented by Waters et al. [WCJ05]. Another example for the utilization of the span-triangle is the isosurface exploration system of Zhang and Kaufman, based on the point generation from voxel cell edges [ZK06].

The span-triangle contains information of potentially active points for a predefined exploration range $[\xi_{min}, \xi_{max}] \subseteq [r_{min}, r_{max}]$. The goal has been the creation of a data structure, which allows an efficient extraction of the explored model as a set of active points for the current target exploration parameter. The major building blocks of the span-triangle are the *base*, *span* and *point data arrays*:

- Each element in the base array corresponds to points with a particular *based* activation minimum value $b = \alpha_{min} - \xi_{min}$ and contains a reference to a span and point data array.

- Each span array is implemented as an offset table, containing indices to the appropriate point information. An item in the offset table corresponds to points with a certain *activation span* $s = \alpha_{max} - \alpha_{min}$.

- During the generation of the span-triangle, all potentially active points stored in the appropriate point data array, ordered by the activation span s. Altogether, points are sorted in the span-space by b and s.

Figure 3.5 illustrates the structure of the span-triangle. The layout of the base and span arrays forms a triangle in the discretized span-space. The base array corresponds to its x-axis and the span array to its y-axis. The point arrays are usually subdivided into smaller memory blocks in order to support robust memory allocation. The span-triangle makes it possible to access active points in a very efficient way, because the extraction procedure can be reduced to a set of linear traversal operations, working on the point information. Let $Base$, $Span$ and $Data$ represent the aforementioned arrays. Furthermore, assume that $Base[b].Span[s]$ returns the offset value to the point information for a based activation minimum value b and activation span s. Let $|Base[b].Data|$ be the size of the $Data$ array, corresponding to the total number of points for b. Algorithm 1 refers to the full extraction routine for the target exploration parameter $[\theta_{min}, \theta_{max}] \subseteq [\xi_{min}, \xi_{max}]$.

Algorithm 1: Full point extraction from the span-triangle.

```
1 for b ← 0 to θ_max − ξ_min do
2 |   s ← max{0, θ_min − ξ_min − b};
3 |   for i ← Base[b].Span[s] to |Base[b].Data| − 1 do
4 |   |   Add information at Base[b].Data[i] to target model;
5 |   end
6 end
```

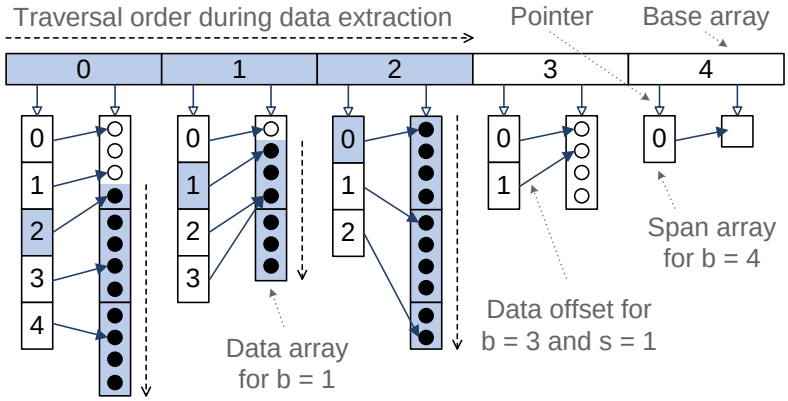


Figure 3.5: Span-triangle data structure for an exploration range of $[\xi_{min}, \xi_{max}] = [0, 4]$ and exploration parameter $[\theta_{min}, \theta_{max}] = [2, 2]$. Active point data is represented by the filled circles. Only array elements, which are colored light blue, need to be accessed.

The algorithm implements the exploration function 3.3 for the interval-based parameter space. The extraction of the target point model is a nonhierarchical procedure and only active points are accessed. It is just a traversal of the base array, combined with look-ups to the point data via the offsets from the span arrays, as shown in figure 3.5. Therefore, the extraction runs in optimal time complexity $O(k)$, where k is the number of active points. The inner loop of the algorithm can be replaced by a memory block operation in order to read out point information with highest efficiency. The *Data* arrays are the dominating part considering the memory consumption. The raw size of the span-triangle, i.e. the number of base and span array elements, is given in equation 3.4.

$$size_{raw} = (\xi_{max} - \xi_{min} + 1) \cdot \left(\frac{\xi_{max} - \xi_{min}}{2} + 2 \right) \tag{3.4}$$

It depends quadratically on the exploration range width $\xi_{max} - \xi_{min}$. Memory restrictions have not to be expected in practice, because the most typical scalar value depth for medical datasets is only 12 bits. Higher values would require a reduction of the scalar space during preprocessing.

Operation Data Structure - Point Grid

The choice of the operation data structure is an application-specific decision. In this section, a simple solution will be presented that can be integrated efficiently into the presented point-based visualization pipeline with respect to fast generation, rendering and data manipulation.

Medical volume datasets are typically "structured" i.e. sample and cell positions can be described as integer vectors with fixed bit lengths. On the one hand, it is therewith possible to derive a compact point representation via quantization of its attributes [Dee95]. On the other hand, only a limited spatial extend is addressable with a few bits per coordinate,

e.g. seven bits result in $(2^7)^3 = 128^3$ *structured points*. A common approach for overcoming such limitations is the utilization of space partitioning, i.e. subdivision of the data into individual buckets. A simple solution is the implementation of a regular grid data structure, i.e. in this case a *point grid*. Each grid bucket is associated with a continuous point data array, effecting in a locally sequential representation. Although such a basic space-driven partitioning does not support adaptivity of the bucketing with respect to the local point density, it makes it possible to realize efficient data insertion and access operations.

The resolution of the grid depends on the spatial extents (dimensions) of the initial volumetric model and the bit lengths of the position coordinates. For example, a 512^3 volume and seven bits per point coordinate result in $(\frac{512}{2^7})^3 = 64$ buckets. In contrast to advanced point compression solutions, which require only a few bits for the complete point position vector, the encoding scheme suggested in this work is very simple [MBAB04, WGE+04, PBCK05, KSW05]. Nevertheless, in combination with out-of-core memory management and the irregular sampling feature of point data, it provides an appropriate representation for large datasets. Moreover, a reasonable trade-off between compactness and rendering performance is achieved, especially with regard to a GPU-based implementation of the decoding procedure.

The point extraction algorithm is based on a straightforward traversal of the grid buckets. Similar to the span-triangle, the array-based representation of the point data makes it possible to read out information with highest efficiency using memory block operations.

3.1.2 Preprocessing

Preprocessing comprises the computation of the core data structures that are part of the point-based visualization pipeline, i.e. exploration and operation data structures.

Generation of the Span-Triangle from the Input Volume

The span-triangle is created in a two-pass approach. In the first pass a *temporary data structure* is generated that includes the extracted point information from the input volume. Afterwards, the exploration data structure is built based on a reorganization of the points for optimal access during interactive exploration. Such a strategy is required for an efficient balance between the speed and memory overhead during the generation: First, a fast radix-sort-based approach with linear time complexity for the sorting of point data by the span-triangle indices b and s can be used [Sed98]. On the one hand, this requires the computation of a *point histogram* during the first pass, including the exact number of points for each possible activation interval. On the other hand, if compared to a single-pass approach, no conservative preallocations of the point data arrays are required. Second, it is possible to avoid additional rearrangement and copy operations of point data during sorting.

The temporary data structure is a single linked list of point memory blocks. The bucketing strategy is required to achieve robust allocation on systems that are prone to memory fragmentation. Each included data element contains the extracted point information and

additionally the indices b and s. The temporary data structure is built by a cell-by-cell traversal of the input volume, similar to the Marching Cubes approach:

1. Generation of a *potentially active* point for each cell, based on the predefined exploration range: A point p is considered as potentially active, if condition 3.2 is satisfied, i.e. $A(p) \cap [\xi_{min}, \xi_{max}] \neq \emptyset$. Otherwise, the point is not inserted into the temporary data structure.

2. Extraction of point attributes from the input volume: Point elements consist of positional information and additional application-dependent attributes. Examples are color, material and gradient information.

3. Computation of point offsets b and s for the base and span arrays: The offsets are computed from the point histogram. This approach is equivalent to the calculation of the counter and offset tables for the radix-sort algorithm. Note that b and s have to be cropped to the exploration range before they can be used for the indexing of the span-triangle, for cases with $[\xi_{min}, \xi_{max}] \subset [r_{min}, r_{max}]$.

A volume cell is represented exactly by one point [CLL+88]. Such an approach can be called *sparse volume sampling* [LT04]. Higher approximation quality can be achieved by on-the-fly supersampling during traversal of the input volume. In the case of a symmetric point primitive, resampling is also required for anisotropic volumes. Therefore, the visualization system described in this work has been originally designed in favor of isotropic datasets, e.g. resulting from cone beam CT reconstruction. Each iteration of the temporary data structure construction needs to access only two volume slices simultaneously. Consequently, with a *slice-based representation* of the volume it is not required to load the complete data into system memory at once (Such an approach exhibits similarities to the *sweep plane* traversal of point data in Pajarola's stream-processing framework [Paj05]). Notice that the entire generation procedure needs to access the input volume only one time. In the case of large volumetric datasets, which have to be stored out-of-core, repeating data access delays can be minimized therewith.

In the second pass of the generation routine, the temporary data structure is traversed and the point information is sorted into the span-triangle using the available point offset tables. There is one special case that has to be considered during this procedure. There may be specific activation ranges that do not occur in the point-based representation of the input volume. Then, the corresponding offset value in the span-array has to be adapted to refer to the next valid point element. Such a situation is illustrated in the point data array for $b = 1$ and $s = 2$ in figure 3.5. As soon as the traversal of a temporary point memory block is finished, its memory is freed. Such a strategy makes it possible to keep an approximately constant number of points in system memory during the whole generation process. Note that the elements of the temporary data structure consume more memory compared to the span-triangle due to the extra storage of the indices b and s. This problem is relativized by an out-of-core bucketing of the temporary data structure. Point buckets are stored in disk memory, as soon as the maximally allowed memory consumption is reached.

Generation of the Point Grid from the Span-Triangle

The point grid is constructed from the span-triangle after fixation of the exploration process. Its generation is a straightforward two-pass procedure. In the first pass, the point counters for each bucket are calculated. These counters are used for preallocation of the data arrays for faster point insertion. In the second pass, the point information from the span-triangle is inserted. Both steps require the execution of algorithm 1 for the fixed exploration parameter. Additionally, it may be necessary to convert the extracted data, if the span-triangle and point grid are based on different point formats. In chapter 4 it will be shown how to use such a simple and non-hierarchical operation data structure for point-based LOD rendering.

3.1.3 Incremental Update

There are two different scenarios for the coupling of the exploration data structure and a rendering component. First, the span-triangle contains data that can be interpreted directly by the renderer, i.e. it is in a hardware-compatible format. Then, the included data arrays can be utilized immediately as input for the rendering algorithm. Second, the included data is compressed or encoded and cannot be processed directly by the renderer. In the latter case, the following point decoding procedure has to be used:

1. **Traversal** of the data in the span-triangle and gathering of relevant information.
2. **Conversion** of the information to a compatible format for the renderer.
3. **Transfer** to the renderer and optionally to graphics memory for display.

Notice that the steps 2 and 3 may introduce a significant delay during exploration. One example is the slow transfer of vertex data from system to graphics memory. Therefore, an important aspect for achieving interactive exploration is the incremental update of the explored model. On every change of the exploration parameter, the currently extracted model is updated without discarding points that are still active. By taking advantage of the coherent layout of the span-triangle, only the necessary information has to be decoded. First, parts of the previously decoded data are freed, which correspond to points that have become inactive after the exploration parameter change. Second, the data for all newly activated points is decoded. The complete exploration process is realized using an incremental version of the full extraction procedure. It is illustrated in algorithm 2, showing the update for a parameter change from $[\theta_{min}, \theta_{max}]$ to $[\theta^{+}_{min}, \theta^{+}_{max}]$.

Notice that the algorithm is presented in a simplified version because of readability. In its optimized form, unchanged portions of the span-triangle do not need to be traversed explicitly and freeing and decoding can be implemented with memory block operations. For a simple example, illustrating the incremental nature of this approach consider figure 3.6.

Another important aspect for preserving interactivity during exploration is the minimization of long continuous decoding times, occurring for large changes of the exploration

Algorithm 2: Incremental point extraction from the span-triangle.

```
1  for b ← 0 to max{θ_max, θ⁺_max} − ξ_min do
2  |  s ← max{0, min{θ_min, θ⁺_min} − ξ_min − b};
3  |  for i ← Base[b].Span[s] to |Base[b].Data| − 1 do
4  |  |  X ← [ξ_min + b, ξ_min + b + s];
5  |  |  if X ∩ [θ⁺_min, θ⁺_max] = ∅ ∧ X ∩ [θ_min, θ_max] ≠ ∅ then
6  |  |  |  Free information at Base[b].Data[i];
7  |  |  else if X ∩ [θ⁺_min, θ⁺_max] ≠ ∅ ∧ X ∩ [θ_min, θ_max] = ∅ then
8  |  |  |  Decode information at Base[b].Data[i];
9  |  |  end
10 |  end
11 end
```

parameter. To handle such cases, a *progressive exploration update* scheme can be incorporated in order to retrieve smooth interaction without blocking the application. During exploration the decoding process is interrupted when a user-defined response time is reached. The decoding and rendering of the remaining point data is then continued on the next frame. More details about progressive refinement will be presented in chapter 4.

3.1.4 Experimental Results

In the following, experimental results are presented regarding the performance and memory consumption of the aforementioned visualization pipeline. In this section, the focus will be on data-processing, including the generation of the exploration and operation data

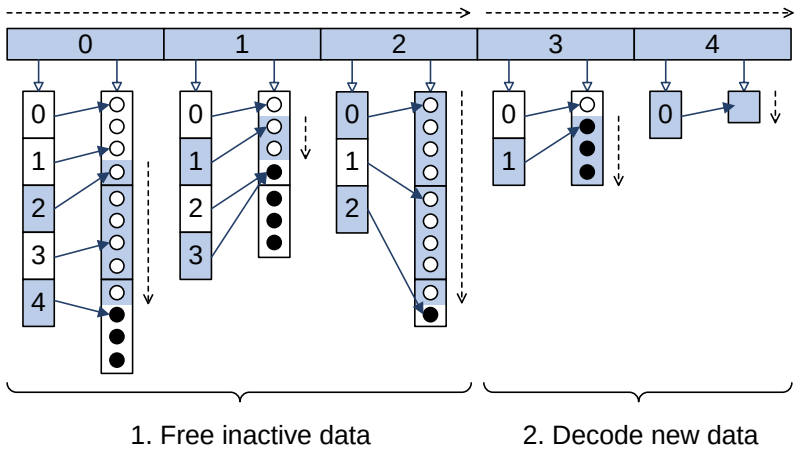


Figure 3.6: Incremental update of the span-triangle for a change of the exploration parameter from $[\theta_{min}, \theta_{max}] = [2, 2]$ to $[\theta^+_{min}, \theta^+_{max}] = [4, 4]$. Only portions of the data structure, which are colored light blue, need to be freed and decoded, respectively.

structures. Rendering-related results and screenshots will be presented in section 3.2.3.

The test computer has been a desktop PC with two AMD64 CPUs at 2.2 GHz, 32 GB of system memory and a NVIDIA Quadro FX 4500 graphics board with 512 MB of memory. The software environment has consisted of the Windows XP Professional x64 Edition operating system with Service Pack 1 and the Microsoft Visual C++ 2005 compiler. All measurements have been executed in single-threaded and 32-bit addressing mode.

Datasets

Table 3.1 gives an overview of the datasets that have been used for evaluation. Mainly CT volumes from the medical domain have been chosen. The *Dry Skull* dataset is a cone-beam scan.

Dataset	Original resolution	Memory	Isotropic resolution	Memory
Head	$320 \times 404 \times 92$	23 MB	$320 \times 404 \times 233$	57 MB
Abdomen	$512 \times 512 \times 174$	87 MB	$512 \times 512 \times 664$	332 MB
Legs	$512 \times 512 \times 500$	250 MB	$600 \times 600 \times 500$	343 MB
Dry Skull	$700 \times 700 \times 700$	654 MB	$700 \times 700 \times 700$	654 MB
Visible Female	$512 \times 512 \times 1734$	867 MB	$512 \times 512 \times 1734$	867 MB

Table 3.1: Volume datasets used for point-based exploration.

All datasets have an allocated scalar depth of 16 bits, utilizing only 12 bits for data-processing. Memory consumption is based on allocated scalar bits. The first three volumes in the table have been resampled in order to achieve isotropy. The associated processing pipeline requires no explicit volume resampling step for this purpose. Instead, the sampling step-size is adapted during traversal of the input volume and generation of the exploration data structure. For information about the originators of the volumetric datasets refer to the acknowledgments at the beginning of this work.

Generation of the Exploration Data Structure

Table 3.2 includes the measurements for the a priori generation of the span-triangle data structure including the full exploration range, i.e. $[\xi_{min}, \xi_{max}] = [0, 4095]$.

Dataset	Subsampling	Time	Points	Memory
Head	122 %	8.9 s	10,448,258	79.7 MB
Abdomen	218 %	8.1 s	9,185,137	70.1 MB
Legs	221 %	10.2 s	13,290,371	101.4 MB
Dry Skull	273 %	11.2 s	14,737,940	112.4 MB
Visible Female	300 %	12.0 s	9,396,977	71.7 MB

Table 3.2: A priori generation of the point-based exploration data structure.

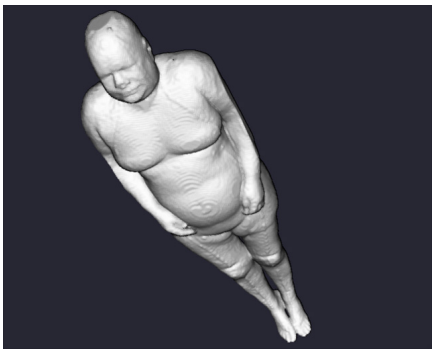


Figure 3.7: Point-based *Visible Female* dataset during a priori exploration.

The subsampling is required to assure short processing times and reduce memory consumption. It is performed on-the-fly, i.e. a resampled volume is not created explicitly. Let d_x, d_y and d_z be the volume dimensions. The subsampling factor s_V is computed automatically with respect to the isotropic volume resolution, such that the initial coordinate dimension is reduced to 256 (equation 3.5).

$$s_V = max\left\{\frac{\sqrt[3]{d_x \cdot d_y \cdot d_z}}{256}, 1\right\} \tag{3.5}$$

Generation times for the exploration data structure include the build up of the temporary data structure, mentioned in section 3.1.2. All resulting timings are reasonable low, i.e. about 10 seconds. Each point in the span-triangle consumes eight bytes of memory. It consists of a quantized position vector with 16 bits for each coordinate and a 16 bits material attribute. Volume cells with constant scalar values, i.e. $x_{min} = x_{max}$, have been omitted during processing. This prevents the generation of points that would not contribute to any isosurface. Similar to most previous point-based isosurface generation approaches, points are arranged regularly at the original volume grid [WM03]. The extension to the irregular case is straight-forward by using floating-point coordinates. Nevertheless, integer-based position vectors provide a significantly more compact representation. The relative low memory consumptions compared to the original volume sizes illustrate the concept of *data reduction via reduction of the input volume resolution*. However, the visual quality of the a priori models is high enough for a first visual inspection, as illustrated in figure 3.7.

Table 3.3 shows the measurements for the a posteriori generation of the exploration data structure.

The exploration ranges have been estimated manually during a priori exploration, targeting at the identification of characteristic anatomical structures, such as skin and bone. The span-triangle generation algorithm is re-executed at full resolution, resulting in higher timing and memory values. Still, the preprocessing times are appropriate for medical practice, because almost all datasets require less than one minute of computation time. For most datasets, the a posteriori generation achieves *data reduction via reduction of the exploration range*.

Dataset	Exploration range	Time	Points	Memory
Head	1000, 1900	6.4 s	8,247,314	62.9 MB
Abdomen	750, 1200	44.1 s	58,365,265	445.3 MB
Legs	2600, 3300	31.6 s	27,949,777	213.2 MB
Dry Skull	1800, 2400	27.4 s	14,292,335	109.0 MB
Visible Female	750, 1400	71.5 s	81,599,419	622.6 MB

Table 3.3: A posteriori generation of the point-based exploration data structure.

The span-triangle data structure supports "random access" to the included geometric information. In the context of isosurfacing, delay times are negligible, providing an interactive experience for the user during exploration. This is a significant advantage compared to previous incremental isosurfacing approaches, which are restricted to small isovalue step-sizes [CVCK03]. As will be shown in section 3.2.3, the total exploration time is mainly determined by the rendering time.

Isosurface Extraction Compared to Marching Cubes

Table 3.4 includes the results for the generation of an isosurface span-triangle compared to the standard Marching Cubes (MC) algorithm [LC87]. The isovalues have been chosen by a medical expert. Notice that the generation of the span-triangle for a single target isovalue is not the typical application scenario. Such a restriction is required here for the comparison between point-based and traditional isosurface extraction.

Dataset	Isovalue	Target	Time	Memory	Time MC	Memory MC
Head	1077	Skin	1.8 s	4.1 MB	2.1 s	8.6 MB
	1730	Bone	1.8 s	4.3 MB	2.1 s	9.2 MB
Abdomen	807	Skin	10.5 s	25.6 MB	6.9 s	32.8 MB
	1125	Bone	10.5 s	25.6 MB	7.6 s	38.8 MB
Legs	2714	Skin	15.8 s	20.6 MB	17.1 s	68.9 MB
	3053	Vessels	17.1 s	37.0 MB	25.3 s	124.2 MB
Dry Skull	1900	Bone	20.5 s	29.2 MB	33.3 s	109.5 MB
	2230	Teeth	18.7 s	4.7 MB	22.1 s	17.5 MB
Visible Female	781	Skin	22.6 s	38.9 MB	44.2 s	145.5 MB
	1260	Bone	21.9 s	30.4 MB	40.5 s	116.5 MB

Table 3.4: Point-based isosurface extraction compared to the Marching Cubes algorithm with quantized coordinates (16 bits) and excluding the computation of normal vectors.

In contrast to the presented point-based processing pipeline that requires isotropic volume data, the Marching Cubes algorithm is executed using the original volume resolutions. The direct processing of anisotropic data is an advantage regarding the performance and memory consumptions, especially in the context of the highly anisotropic volume *Abdomen*.

The Marching Cubes generates a triangle mesh. The main factors regarding the memory consumption are the corresponding vertex and normals arrays as well as the triangle index list. In the traditional version of the algorithm, each vertex and normal vector is represented by three floating-point values. The vertex indices are encoded as integer values with 32 bits. For a fair comparison to the described point-based representation, triangle vertices have been assumed to be quantized to 16 bits. Additionally, normal vector computations have been not considered in the table. The standard Marching Cubes requires ca. 20 % more processing time and ca. 60 % more memory in practice without these "optimizations".

Moreover, only a part of the traditional isosurface workflow has been considered. Usually, additional processing steps are necessary, like triangle mesh decimation and smoothing. These can additionally reduce the rendering time and memory costs, but introduce a significant preprocessing time overhead and therewith deranging delays for the user. The developed point-based pipeline usually requires no additional computations in order to achieve a rendering-efficient data representation.

The point generation times are mainly determined by the original volume resolutions. Most time is spent on the identification of potentially active isosurface cells. The creation of one point for each volume cell effects in a lower geometric primitive count compared to the Marching Cubes. It is a benefit regarding the memory efficiency, but usually comes along with a degradation of visual quality. Please refer to section 3.2.3 for more details on this topic. Additional benefits regarding memory consumption are the reduction of the exploration range, irregular sampling of the input volume and the utilization of compact point formats. Noisy datasets, such as *Legs*, result in an increased number of primitives. For such data, smoothing of the input volume during preprocessing might be necessary.

Generation of the Operation Data Structure

Table 3.5 shows the results for the generation of the operation data structure from the span-triangle after fixation of the exploration parameter. Included are measurements for isosurfaces, extended surfaces (*Legs*, *Dry Skull* and *Visible Female*) as well as for full volumetric representations (*Head* and *Abdomen*).

The bucket structure of the point grid makes it possible to take advantage of a different point format compared to the span-triangle. Each primitive is twice as compact, i.e. it is encoded as a 32 bits integer value with seven bits for each positional coordinate and 11 bits for the material value. For datasets with large scalar widths, i.e. extended surfaces and full volumetric datasets, one can observe a significant increase of memory consumption. For *Head* and *Abdomen*, the consumption is even higher than the size of the initial anisotropic volumes. In spite of encoding of each point using four bytes, the original volume datasets require only two bytes per sample. Therefore, point-based models are more efficient only in cases where one can take advantage of their irregular sampling feature, i.e. surfaces and extended surfaces.

The generation of the operation data structure requires just a few seconds for smaller scalar ranges and 1-2 minutes, otherwise. One important factor for its fast computation is the aforementioned support for "random access" to the span-triangle.

Dataset	Scalar range	Time	Points	Memory
Head	1077, 1077	0.5 s	541,620	2.1 MB
	0, 4095	19.5 s	29,018,772	110.7 MB
Abdomen	1125, 1125	2.8 s	3,351,689	12.8 MB
	0, 4095	109.6 s	170,625,600	650.9 MB
Legs	2714, 2714	2.5 s	3,043,463	11.6 MB
	2714, 4095	18.0 s	28,061,756	107.0 MB
Dry Skull	1900, 1900	3.0 s	3,829,973	14.6 MB
	1900, 4095	5.8 s	8,974,213	34.2 MB
Visible Female	781, 781	4.4 s	5,094,015	19.4 MB
	781, 4095	52.7 s	81,856,959	312.3 MB

Table 3.5: Extraction of the point grid from the span-triangle.

3.2 Universal Rendering

One goal of this work has been the design of a highly configurable and efficient point-based visualization technique that is appropriate for maximal utilization of graphics hardware acceleration. The associated renderer is implemented as a single component, capable of producing various types of 2D and 3D medical visualizations, just by modifying its input parameters. Such an approach can be considered as the conceptual counterpart of Hauser's *two-level volume rendering* (2lVR) that displays distinct portions of the volume dataset with individual rendering techniques [HMBG01]. Moreover, by mixing different parameter configurations, the component in this work is capable of producing hybrid visualizations, e.g. by mixing surface and volumetric representations. It should be mentioned that due to the high complexity of medical models, consisting of several millions of points, the main focus has been on high throughput instead of high visual quality. The *splatting* rendering technique has been chosen, i.e. an object-order approach that can be incorporated efficiently in a graphics-hardware-oriented pipeline (section 2.1).

With reference to section 3.1.1, the main principles in the context of interactive medical applications, which have influenced the design of the renderer in this work, are as follows:

Universal - The renderer should support various types of point data structures, which can be represented as *collections of point arrays*. Ajar to the concept of the universal point primitive, it should be possible to create different visualizations with one component, e.g. for surfaces and volumes.

Single-pass - In order to achieve high rendering performance the renderer should traverse its input data only once per rendering frame [ZP06]. Such a single pass approach, with respect to the point data, is essential for the simple integration of efficient progressive refinement, as will be described in section 4.1.

Order-independent - The traversal of the input primitives should be not dictated by the renderer, i.e. it should be order-independent. The background of this requirement is

the utilization of the point order for level-of-detail management, as will be presented in section 4.2.

The aforementioned principles are summarized in the high-level overview of the universal point renderer, illustrated in algorithm 3. The traversal of the input point data structure for each object is realized with an abstract *point iterator* interface, resulting in a sequential and array-wise processing of the input points. Therefore, the developed rendering technique can be applied to all point-based data structures, which satisfy the design principles of section 3.1.1. The span-triangle and the point grid are just two examples. More details, including the extended framebuffer and splat information formats, will be described in the remainder of this section. Moreover, the presented algorithm will be adapted in section 4.2.2 in order to reduce the rasterization overhead during rendering of large splats, resulting from lower levels of detail and high magnification.

Algorithm 3: High-level overview of the universal point rendering algorithm.

```
1  foreach point object O do                                  /* Splatting step */
2  |  foreach point array P ∈ O do
3  |  |  foreach point p ∈ P do
4  |  |  |  Project p to image plane point p';
5  |  |  |  Compute splat information s from p';
6  |  |  |  Write s to extended frame buffer F;
7  |  |  end
8  |  end
9  end
10 foreach pixel i ∈ F do                                     /* Shading step */
11 |  Estimate normal vector n using neighbors of i;
12 |  Compute shaded color c by applying n;
13 |  Write c to final image buffer;
14 end
```

Deferred Splat Shading

The *1+1 renderer* described in this work is based on two separated steps: visibility (splatting) and illumination computations (shading). This results in a *deferred shading* system, sometimes called also *quad shading* [ZP06, Fah06]. In contrast to traditional forward shading approaches, lighting computations are omitted during rasterization of the scene geometry. Instead, objects are rendered to an extended framebuffer with additional geometric and material information, usually called *geometric buffer* (known synonyms are also *fat framebuffer* and *2.5D image space*) [Har04]. Then, shading computations are performed pixel-wise in 2D as a postprocessing step.

Deferred shading was introduced 1988 by Deering et al. in the context of a hardware architecture that was based on the separation between triangle rasterization and shading [DWS+88]. Two years later, Saito et al. presented the concept of geometric buffers (G-buffers) for storing intermediate results during rendering, including position and normal vectors, material attributes etc. [ST90] Driven by the computer game industry of the last years, deferred shading has become also practical in the context of real-time rendering on commodity graphics hardware. Various GPU-based solutions have been developed, supported by the evolution of shader programming [EWE04, PF04, KK05].

The application of deferred shading is very attractive for high-performance rendering, because shading computations are only executed for visible pixels. Additionally, the G-buffer can be used as the input for custom image postprocessing effects, like smoothing, edge detection and depth attenuation (i.e. "fog"). But there are some challenges that have to be kept in mind during the renderer design, like the high memory intensity due to the extended framebuffer format and the requirement of modern GPU features.

Typical deferred shading techniques work with precomputed normals that have to be stored in the G-buffer. Additionally, the utilization of complex splat shapes requires extra per-point information, such as elliptical tangent axes [BHZK05]. Such approaches result in a complex framebuffer format and usually require the utilization of multiple render targets. The approach presented in this work stores only simple per-splat information and normals are computed on-the-fly in the deferred step. The corresponding rendering solution is called *deferred splat shading*, working on the *geometric splat buffer* (GS-buffer).

3.2.1 Splatting

The first step of the presented rendering algorithm is very similar to traditional *visibility splatting*, i.e. the determination of the visible splat pixels [RPZ02]. Furthermore, for each pixel in the GS-buffer splat-related information is stored, including the associated color, splat size and visual order. In the following it will be described how to utilize such a simple data format for the generation of the final image. There is a separation between *per-splat* and *per-splat-pixel* operations, making it possible to map the complete splatting step directly to the vertex and fragment stages of modern computer graphic cards. One important goal is high performance by minimizing the number of operations in each stage. The resulting challenge is the creation of meaningful visualizations at adequate visual quality in spite of the simple calculations.

Per-splat operations

In the following an overview is given of the operations that are performed by the renderer for each point and resulting splat, respectively. Calculations are separated between geometry, material and order:

Geometry - Each input point may be encoded in a compact format. An example is the quantization of position and material values (section 3.1.4). For such cases, the

first rendering step includes the decoding of the point attributes directly on the GPU. The next step is the transformation and clipping of coordinates, including the computation of the projected splat size. Additionally, a *fat clip plane* can be specified as follows: Let $f(\vec{x}) = \vec{n} \cdot (\vec{x} - \vec{x}_0)$ be the clip plane equation for the normal vector $\vec{n}$ and plane point $\vec{x}_0$. Let t be the plane thickness. Then, a point $\vec{p}$ gets clipped, if the following condition is satisfied:

$$f(\vec{p}) > 0 \vee f(\vec{p}) + t \leq 0 \tag{3.6}$$

Material - A transfer function is used to map material identifiers to RGBα color values. The mapping is implemented using a look-up table and texture mapping hardware. Notice that this kind of operation could be performed also in the deferred shading step. Nevertheless, such an approach would complicate the specification of material identifiers, requiring a global material management for the whole scene. Simple "per-object materials" have been preferred, because such a solution is more common in medical visualization applications.

Order - The order for each splat is specified as follows: Let z be the normalized splat distance to the image plane, c_α the normalized alpha component of the associated color value and $w \in [0, 1]$ a user-defined *order weight*. Then, the final order key is determined by the following linear formula:

$$z \cdot (1 - w) + (1 - c_\alpha) \cdot w \tag{3.7}$$

Similar to Heidrich's et al. rendering technique, the order key is used to determine the visibility using standard z-buffer hardware [HMS95]. A splat pixel passes the test, if the incoming order value is less than the stored value. The developed renderer can be utilized therewith for various types of visualization. Setting $w = 0$ results in ordinary opaque rendering, $w = 1$ in maximum-intensity-projection (MIP) volume rendering and $w \in (0, 1)$ in *hybrid visualization* (see page 55 for more information).

In the remainder of this section more details are provided regarding the generation of point-based MIP and related visualizations, including a short overview of previous methods.

Point-based MIP visualization

The order-independent renderer design principle, associated with the high performance goal in this work, has no negative impact on the rendering of opaque surfaces. For optically realistic ("composite") volume rendering there is another situation, because there exists no efficient order-independent solution. Everitt's graphics-hardware-oriented *depth peeling* technique is not practical for objects that exhibit a high depth complexity [Eve01, GBP06]. Similarly, Zhang's and Pajarola's algorithm handles transparency efficiently only for a few surface layers [ZP06]. Therefore, the focus is on MIP, an order-independent variation of volume rendering.

Figure 3.8: Point-based maximum-intensity-projection of the *Abdomen* dataset.

In contrast to a sorted compositing of optical properties along the viewing ray, the classic MIP method selects the volume sample with the maximum scalar value for each image pixel. Structures with high intensities occlude structures with lower values (figure 3.8). Therefore, intensity can be interpreted as an importance or order key for each volumetric primitive. The resulting disadvantages are missing shading, depth and occlusion information. Nevertheless, MIP has proven to be useful in medical applications, especially for the visualization of bony structures and blood vessels from CT and MR angiography, respectively [PP03, VBB+06]. Moreover, most medical experts are familiar with such order-independent techniques, reminding them of the traditional X-ray display. In addition, some researchers argue that they are easier to control - in contrast to realistic optical models - and have the potential to reduce the information degeneration problem due to occlusion of thin interior structures [ME05].

MIP rendering can be implemented in a straight-forward manner using the raycasting approach and searching for the volume sample with the highest intensity value for each ray. It is hard to achieve interactivity for a brute-force implementation, because all scalar values have to be evaluated. Therefore, many acceleration techniques have been developed. The first example is Sakas' et al. raytracer, which performs trilinear interpolation only for relevant voxels along the ray, based on a maximum cell intensity check [SGS95]. Heidrich's et al. MIP renderer works for multiple semi-transparent isosurfaces [HMS95]. By applying a clever transformation for each triangle mesh before rasterization, the authors can take advantage of z-buffer hardware for the implementation of the maximum test. Mroz et al. have accelerated MIP visualization by applying a linear data reorganization technique, including skipping of empty regions [MGK99]. Relevant voxel cells are stored in an ordered array, sorted by their maximum scalar values. By traversing only a prefix of the array it is possible to generate an approximated preview image. A multiresolution technique has been presented by Kim et al. [KP01]. Their solution is based on a dual representation of the volume data. An undersampled representation is used for the generation of a fast preview image. The resulting low-resolution MIP is used to speed up the computation of the final image. Shareef et al. as well as Mora et al. have presented two GPU-based solutions for MIP rendering using 3D texture-mapping and blending hardware [SC01, ME04].

Pekar et al. have revisited the raycasting approach [PHK+03]. The input volume is divided into blocks, each one associated with a precomputed maximum scalar value. The blocks are ordered according to their maximum values along the viewing ray, making an efficient skipping of empty space and invalid regions possible. Finally, Mora et al. have further developed MIP raycasting by utilizing an octree data structure and gray-level quantization [ME05].

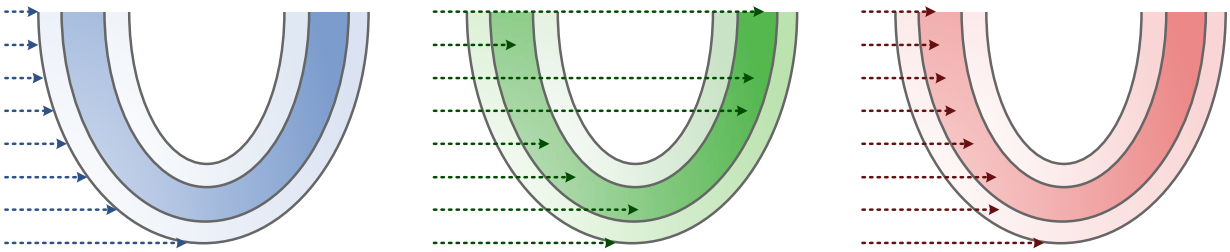
Nearly all techniques are purely software-based, making interactive visualization of large datasets a hard challenge. Moreover, memory-demanding and complex data structures are also potential sources of problems. The point-based approach in this work combines various advantages in one system. One example is the high efficiency due to skipping of empty regions and full graphics hardware acceleration. Another example is the handling of complex datasets via progressive refinement and level-of-detail rendering, as will be presented in chapter 4.

MIP extensions

Mixing opaque surface and MIP volume rendering results in a hybrid visualization technique, just by using a linear combination of the depth and intensity for the z-buffer test. In this work, the intensity is represented by the alpha color component, effecting in more flexibility via transfer function control. The effect of the presented hybrid visualization is illustrated in figure 3.9. It can be utilized to emphasize structures of high intensity (i.e. alpha value), which are located near the object surface along a viewing ray. Additionally, shading can be enabled to highlight edges in the hybrid image. It has been observed that this may improve the perception of complex structures only for low w values in formula 3.7. For other order weights, the resulting images appear less intuitive. On the one hand, the non-photorealistic visualization appearance is a by-product of the universal rendering technique and simple to control. It requires no dual data representation, extra preprocessing and explicit management of feature information. On the other hand, much more sophisticated techniques for illustrative volume visualization are existing, including silhouette enhancement, feature lines, context-preserving rendering and compositing of different rendering modes [YC04, TIP05, VGH+05, KSW06]. An example of the point-based hybrid rendering techniques is shown in figure 3.10.

The problem of missing depth information in traditional MIP rendering is reduced by the motion parallax effect during animation and interaction. A more tangible solution is to provide "depth cues" for the viewer and to extend the MIP to a *depth-shaded MIP*, called DMIP [HMS95, SC01]. The corresponding idea is the same as for "fog rendering" and "distance color blending". A point that is further away from the current viewer position is associated with a lower intensity and the background color, respectively [RE01]. The depth attenuation of intensity values is implemented as follows: Let $d(z) \in [0, 1]$ correspond to a "distance weight function", e.g. a linearly decreasing function with respect to the viewing distance z. Based on formula 3.7, the final order key is determined using:

$$z \cdot (1 - w) + [1 - c_\alpha \cdot d(z)] \cdot w \tag{3.8}$$

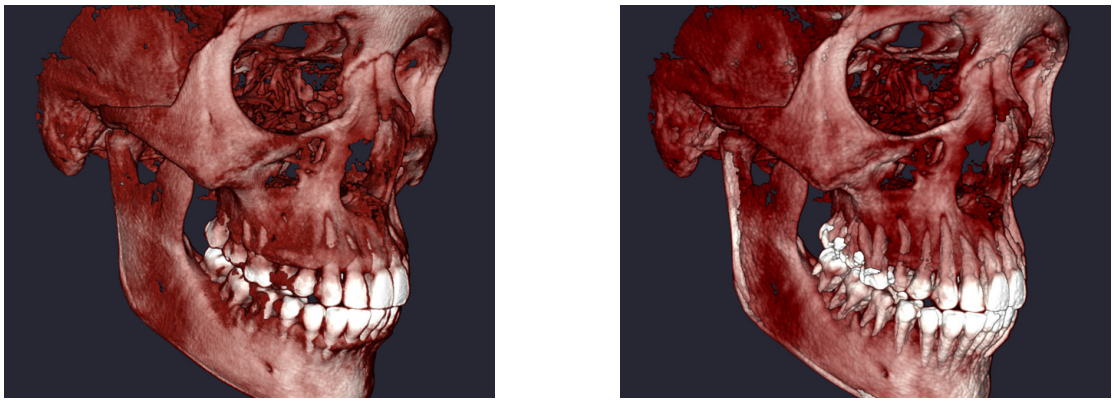


(a) Surface rendering (b) MIP rendering (c) Hybrid rendering

Figure 3.9: Illustration of point-based hybrid visualization using viewing rays. The oval shape represents a mandible. Dark colors show structures with high intensity values (teeth). Bright colors illustrate structures with lower intensities (mandibular bone).

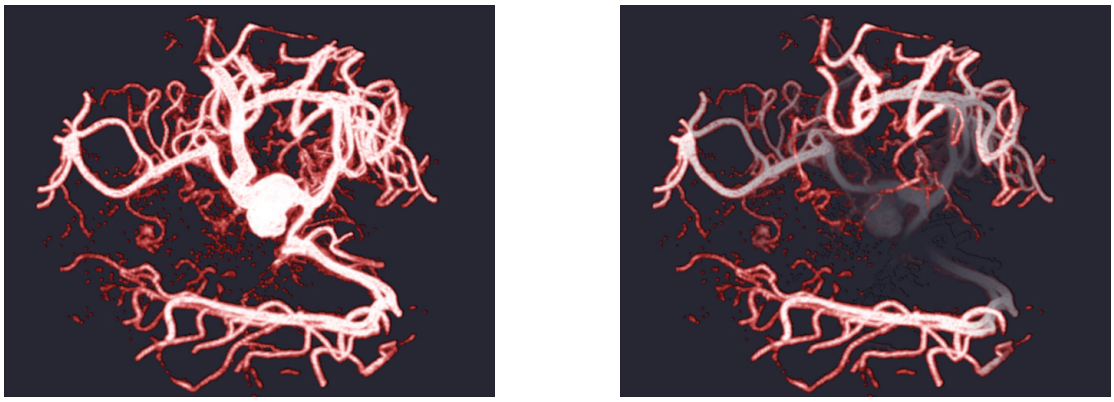
Similarly, the distance weight function can also be utilized for blending between the current and the background color, i.e.: $c_{current} \cdot (1 - d(z)) + c_{back} \cdot d(z)$. The comparison between a point-based MIP and DMIP visualization is illustrated in figure 3.11.

The final extension is motivated by the huge number of slices in modern radiological datasets. Displaying and investigating each slice separately may be not applicable in clinical practice. Therefore, Napin et al. have introduced the *sliding-thin-slab MIP* (STS-MIP) technique [NRJ93]. Their main contribution is the computation of the MIP for a slice "slab". It can be considered as a "fat slice", consisting of multiple images and effecting in faster sliding through the volume data. In this work, STS-MIP is interpreted as a degenerated 3D visualization. Based on condition 3.6 and formula 3.8, it can be simulated by the universal renderer just by enabling a thin clip plane and MIP functionality (figure 3.12).



(a) Low order weight (b) Exposed dental roots

Figure 3.10: Point-based hybrid visualizations with enabled shading.



(a) MIP (b) DMIP

Figure 3.11: Point-based MIP and DMIP visualizations of brain vessels.

Per-splat-pixel operations

In the following, the per-pixel operations that have to be executed for each splat are described, including the calculation of the splat shape and the final GS-buffer write:

Splat shape - Due to the utilization of symmetric point primitives and the high rendering efficiency requirement, only rectangular and circular splats are considered [RL00]. As mentioned by Zwicker et al., such simple and piecewise constant shapes require no perspective correction [ZRB+04]. Another advantage is the potential acceleration of the rendering by a factor three to four compared to the relative complex surfel primitive, as reported by Guennebaud et al. [GBP04a]. In case of rectangular splats, no additional operations are required. For circular shapes, a test for outside lying pixels is required using a straight-forward pixel-to-splat-center distance check.

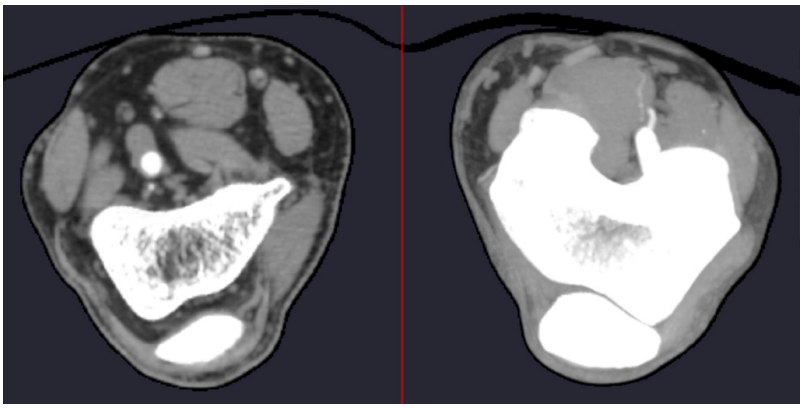
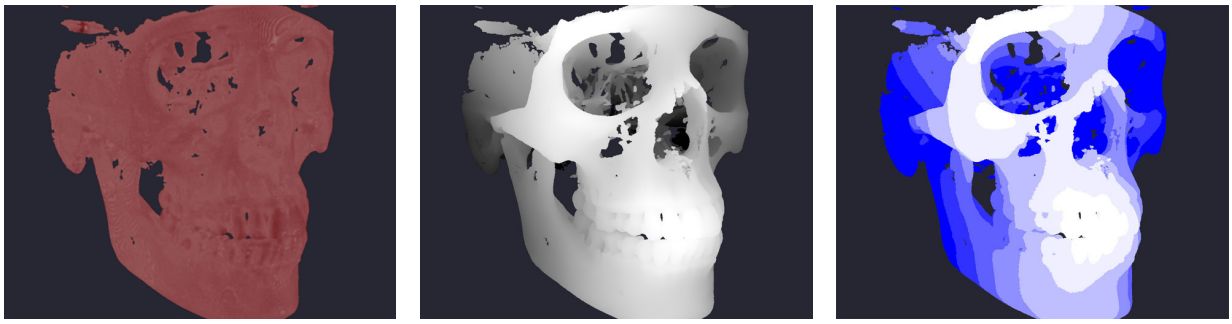


Figure 3.12: Point-based STS-MIPs of a human leg dataset. The left image shows a single slice. The right image illustrates the effect of a thick STS-MIP, consisting of multiple slices.



(a) Color (b) Order (depth) (c) Splat size

Figure 3.13: Splat information after first rendering step. The order values are colored from white (near) to black (far). The splat sizes are colored from white (large) to blue (small).

GS-buffer write - For each pixel, the data that has been computed during the per-splat stage is written to the GS-buffer, i.e. splat size, RGB color and order key. For a GPU-based implementation, the GS-buffer can be represented by a floating-point texture with three components. Therefore, extra computations are required to map between the RGB color and a floating point value. The order key is passed to the z-buffer test. Finally, the GS-buffer is used as input for the subsequent shading step.

Although the majority of the presented calculations is associated with the per-splat phase, the actual mapping to the GPU's vertex and shader units may exhibit another structure. The intrinsic distribution of the corresponding operations is influenced strongly by shader code optimizations as well as hardware and driver limitations. For example, the implementation of the transfer function look-up and the calculation of the order key are executed in the fragment stage. One reason is the relative slow vertex unit texture-access of current graphic boards. Another reason is the goal for a more balanced allocation of the vertex and highly parallel fragment operations. The results from the first step of the universal rendering are illustrated in figure 3.13, including the color, order and splat size values. Notice that the information content of figures 3.13(b) and 3.13(c) is similar. The order (i.e. depth) values could be used to compute the splat sizes on-the-fly. Nevertheless, there are three reasons that prevent such an approach: First, it is applicable only in case of surface rendering. Second, it would require additional transformation operations for each pixel, effecting in a significant slow-down of the shading calculations. The final factor is the visualization with adaptive point sizes that will be presented in the next chapter.

3.2.2 Shading

The second step of the universal rendering algorithm is independent of the complexity of the input model, i.e. it is performed in pixel space. Its main task is the estimation of the normal vector for shading calculations at each pixel, using the splat information from the previous step. The result is the final image, representing the entire point-based scene. All

developed shading techniques can be applied directly to non-point-based rendering, but this is not the focus of this work.

The visual quality of surface shading is highly dependent on the normal estimation method. The approach in this work is based on a decomposition of the currently visible scene parts into *microtriangles*. Then, computing the normal vector, i.e. the *micronormal*, becomes a straight-forward task. Such an approach has parallels to the idea of physically-based shading models, based on *microfacets*. There, surfaces are modeled as collections of microscopic facets, each one representing a specular reflector [CT81]. There are also similarities to the *Reyes* rendering architecture. The *dicing* operation disassembles objects into *micropolygons* [CCC87]. Then, these are used to produce various highly detailed shading effects, like displacement mapping. In the remainder of this section, an overview of related normal estimation approaches will be provided. After that, the technique for point-based models will be presented, which takes care of possible surface discontinuities and has been implemented entirely on the GPU.

Normal estimation

Traditional normal estimation techniques in medical visualization are performed during preprocessing, utilizing a 3D operator in voxel space (*gray-level gradient shading*) or by fitting a plane or polynomial patch to the previously identified surface voxels (*contextual shading*) [Lev88, YCK92, Car96, NCKG00, DLS05]. Generally, such techniques produce high-quality results, but imply an additional memory overhead due to the explicit storage of normal information. Additionally, costly operations are usually required to update the normal vectors on viewing or geometrical changes of the data, e.g. after virtual bone cutting. One example is the *reversed gradient method* by Jezek et al., i.e. the flipping of normal directions for all visible object backfaces [JA03].

The technique presented in this work is based on the computation of normals from the depth values, resulting in the so-called *z-buffer shading* and *depth gradient shading*, respectively. Such an approach facilitates the development of compact point data structures and requires no normal recalculation after model manipulation. As illustrated in figure 3.14, it is possible to provide therewith a consistent visualization, also for dynamically changing geometry.

The most popular depth normal techniques are based on the pixel-space approximation of the gradient vector $\vec{n} = (\frac{dZ}{dX}, \frac{dZ}{dY}, -1)$ by calculating differences between the associated depth values. The most common operators are the backward, forward and central differences (figure 3.15). Usually, the quality of the reconstructed normals is rather low, as reported by Yagel et al. [YCK92]. Typically, the resulting normals have a low resolution and an uneven distribution. Another source of problems is the high sensitivity to discretization artifacts and noise as well as "dark bands" in the image, resulting from discontinuities in the depth map (e.g. cliffs and overhangs). Nevertheless, the basic difference operators are popular in the context of interactive rendering due to their simplicity and efficiency. Already in 1998, Westermann et al. used basic frame buffer arithmetics to compute forward differences for shading in their 3D-texture-mapping volume renderer [WE98].

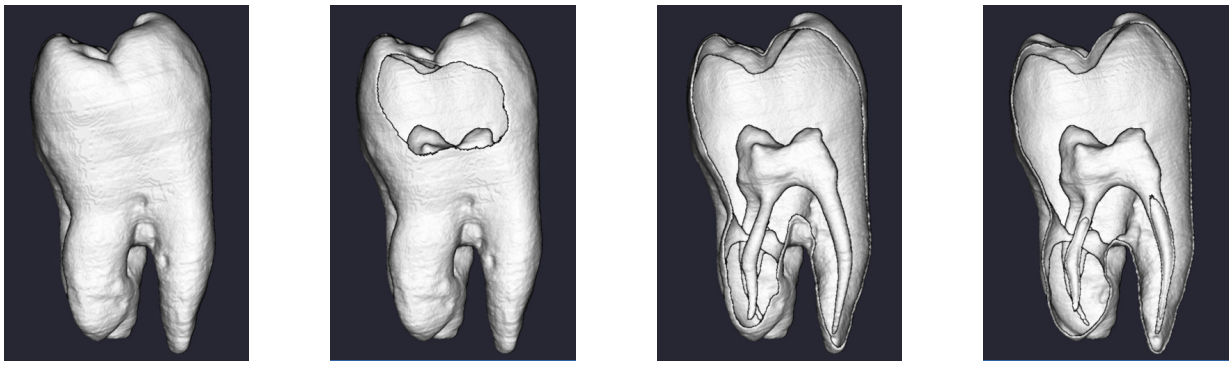


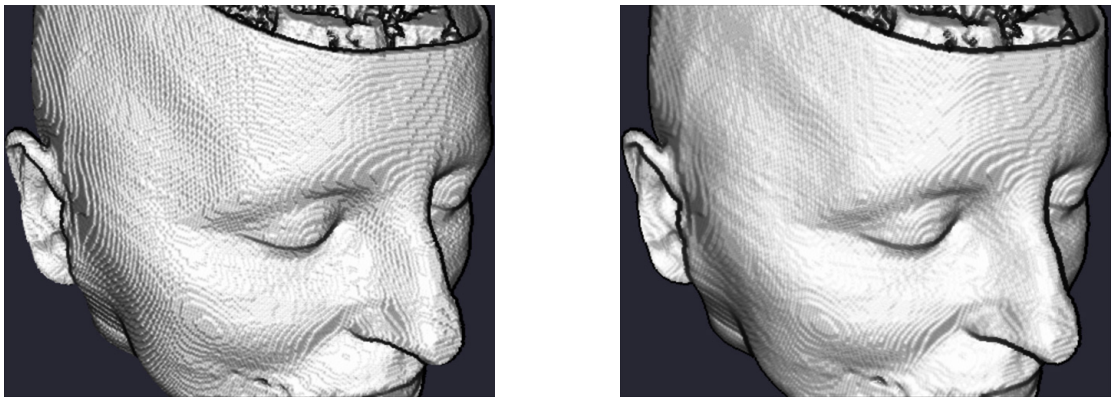
Figure 3.14: Clip plane sequence of a tooth isosurface model.

Difference-based approximation techniques have been further developed to improve the normal estimation quality, mainly by detecting C0 discontinuities in the depth image using a depth threshold check. Examples are the contributions of Gordon and Tiede et al., which are based on a weighted sum of backward and forward differences [GR85, THB+90]. Larger differences are considered as potential discontinuities and are therefore associated with smaller weight values. Another approach, inspired by image-processing and edge-detection applications, has been presented by Magnusson et al. [MLD88]. The authors have proposed to use the Sobel filter to compute the gradient magnitude for z-buffer shading. They report similar visual artifacts, especially the darks bands.

In the year 2004, Kawata et al. picked up again the work on depth gradient estimation techniques. The first example, realized in their image-based point rendering system, is based on a construction of small faces using the depth values of neighboring pixels [KK04]. The final normal vector is computed by averaging multiple face normals at each pixel. On the one hand, some aspects are similar to the microtriangle approach in this work. On the other hand, Kawata's et al. solution is based on a complex face construction, requiring the consideration of 24 different patterns. Additionally, discontinuity is detected using only a depth threshold check. Like for all aforementioned techniques, the size of their normal estimation operator is fixed to 3×3 pixels and does not consider varying splat sizes in pixel space. Their software-based implementation achieves only a few frames per second. One year later, the same authors developed a GPU-based normal estimation technique based on a plane fitting in z-buffer space [KK05]. Their approach is slowed down by the solution of a linear equation system for each pixel, e.g. using the Gaussian elimination method. The handling of discontinuity artifacts is not addressed.

Micronormals averaging

The overview of related z-buffer normal estimation techniques reflects that the main deranging problems regarding visual quality are the dark band artifacts and a noisy image appearance. The origin of the former problem is the lack of surface boundary information in the depth image. As illustrated in figure 3.15, discontinuities can be interpreted falsely



(a) Forward differences (b) Central differences

Figure 3.15: Image-based normal estimation, including noise and dark band artifacts.

as steep gradients, effecting in "thick edges" [CGPD98]. The main reason for the latter problem is the utilization of only a few depth samples for the computations. Additionally, early methods had to tackle limitations resulting from low-precision z-buffers. Nowadays, modern graphics hardware provides data buffers and calculations at floating-point precision. To overcome the aforementioned limitations the *micronormals averaging* technique has been developed. It can be considered as a compromise between high image quality and fast computations, which can be mapped efficiently to current GPUs.

In the following, the contents of the framebuffer are interpreted as a depth map $D_{i,j}$, containing 3D points in the world coordinate system. The mapping from a 2D pixel position and z-buffer value to the world coordinate system can be accomplished using the inverse viewing transformation. The same method can be utilized for the reconstruction of the pixel splat size $s_{i,j} \geq 0$ in world coordinates, i.e. $s'_{i,j}$. For each pixel (i, j) the following operations have to be performed:

1. **Microtriangles** - The first step comprises the construction of eight microtriangles. Each microtriangle shares the current depth map point $\vec{v}_0 = D_{i,j}$ as a common vertex. Its other two vertices $\vec{v}_k$ and $\vec{v}_l$ (with $1 \leq k, l \leq 8$) are identified using horizontal, vertical or diagonal neighbor points. In contrast to Cook's Reyes rendering architecture, that is based on micropolygons at the size of a pixel, the distance to the microtriangle neighbors is dictated by the current splat size information $s_{i,j}$ in the GS-buffer. Therefore, each neighbor vertex is represented by a depth map point $D_{i+\delta i,j+\delta j}$ with $\delta i, \delta j \in \{0, \pm s_{i,j} \cdot c_s\}$. c_s is a user-defined *splat size factor*, used to control the smoothness of the estimated normals. The geometrical adaptivity of the normal estimation operator is required to compensate the constant splat structure, as shown in figure 3.16.

2. **Selection** - In the second step, all neighboring microtriangle vertices are checked for continuity with respect to the current depth map point. Let z_m be the z-component

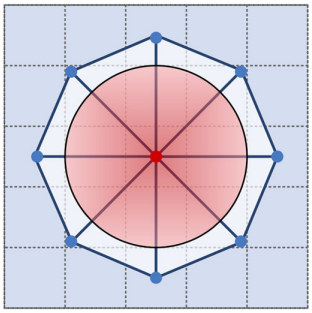


Figure 3.16: 2D representation of the normal estimation operator. Image plane pixels are represented by the blue background grid. The large red circle illustrates a rendered splat, having its center at the red pixel sample. Neighboring samples are used to construct the microtriangles and are shown as blue dots. The operator has an octagonal form.

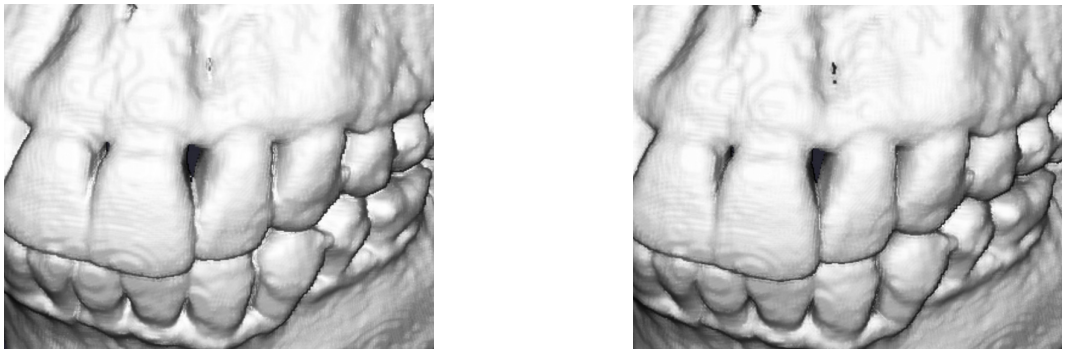
of a vertex $\vec{v}_m$ with $1 \leq m \leq 8$ and $\bar{d} = \frac{1}{8}\sum_{n=1}^{8} |z_n - z_0|$, i.e. the average depth difference. A microtriangle with vertices $\vec{v}_0$, $\vec{v}_k$, $\vec{v}_l$ and a maximal depth difference $d_{max} = max\{|z_k - z_0|, |z_l - z_0|\}$ is *selected*, if the following condition is true:

$$\frac{d_{max}}{s'_{i,j} \cdot c_s} \leq c_0 \quad \vee \quad \frac{|d_{max} - \bar{d}|}{\bar{d}} \leq c_1 \tag{3.9}$$

The left part corresponds to a conservative C0 continuity check based on the depth values of the current microtriangle. The right comparison is a test for the relative error with respect to the average depth value. It is used to counterbalance the condition in regions of the depth map with a steep gradient, assuring a smooth image appearance. Altogether, a microtriangle is selected if it passes the depth continuity check or if the corresponding relative error is small. The microtriangle selection can be interpreted as the *context segmentation* step in Yagel's normal estimation framework [YCK92].

3. **Micronormals** - In this step the normalized micronormals are computed, i.e. $\vec{n}_m = |(\vec{v}_k - \vec{v}_0) \times (\vec{v}_l - \vec{v}_0)|$ with $m \leq M$, where M is the number of previously selected microtriangles. This normal computation procedure reveals some conceptual similarities to the technique, implemented in Kadosh's et al. surface raycaster [KCoSY99]. There, three adjacent ray-surface intersection points that lie within a predefined distance range are used to define a plane for the calculation of a per-pixel normal vector.

4. **Averaging** - In the last step, the final normal $\vec{n}$ is computed as an average of all available micronormals. One possibility is to use simple and fast arithmetic averaging. The associated summation of 3D direction vectors may result in additional visual artifacts, typically observable as "bright bands" near valley regions of the depth image. Therefore, it is suggested to perform the averaging in the spherical coordinate system (figure 3.17). Then, additional trigonometric functions have to be executed for



(a) Arithmetic (b) Spherical

Figure 3.17: Different strategies for averaging of micronormals.

each micronormal, effecting in a noticeable slow-down of the whole normal estimation procedure.

It has been observed that the following parameter settings effect in an adequate visual quality: $c_s \in [\frac{1}{2}, 1]$, $c_0 = \sqrt{3}$ and $c_1 = 1$. The default value of c_0 corresponds to the maximal distance between samples in a regular volume grid. If no micronormals have been selected, the final normal vector is undefined at the corresponding pixel position. In order to preserve a smooth image appearance it is suggested to ignore the selection process for such cases and to apply the overall average normal vector. As presented in figure 3.18, the number of undefined normals is rather low for a typical dataset.

Color filtering and shading calculations

The next step of the shading pass is the smoothing of the RGB colors in the GS-buffer. This process can be considered as a texture-filtering operation with the following extensions: First, the deferred shading approach results in the filtering of visible color information only. Second, the filtering operation is geometrically adaptive, i.e. the size of the

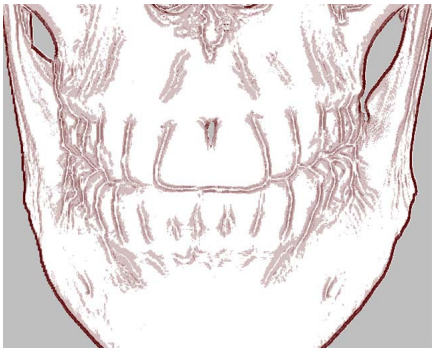


Figure 3.18: Selection of microtriangles, colored from red (low count) to white (high count).

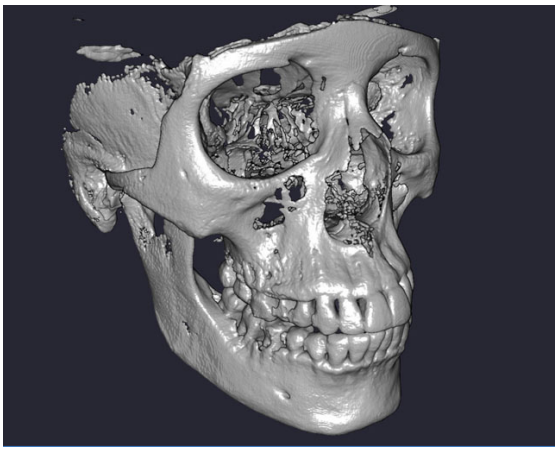


Figure 3.19: Point-based surface visualization with specular highlighting.

filter operator is derived from the splat size information in the GS-buffer. Similar to the micronormals estimation technique, a context-sensitive averaging of neighboring colors if performed with respect to the aforementioned continuity check. Only image points, which satisfy condition 3.9, contribute to the color averaging. As mentioned in section 3.2.1, each access to the splat information in the GS-buffer requires the decoding of color information. In this context, advantage cannot be taken of standard texture interpolation functionality. Instead, bilinear interpolations of geometric and material data have to be performed directly in the shader.

The lighting intensity at each pixel is computed with the Phong shading model [Pho75]. Although the described rendering algorithm provides only *per-splat-exact shading*, the combination of normal and color averaging with the highly irregular structure of the depth map results in a Phong-like shading appearance (figure 3.19). Moreover, the utilization of one light source - located at the eye position - is sufficient for most medical visualization applications. Advanced shading effects, like shadows and reflections, can be considered as "natural artifacts", reducing the information value of the rendered image for medical users.

3.2.3 Experimental Results

In this section, the results for the universal rendering technique regarding performance and visual quality will be presented. Included are comparisons to the traditional visualization approaches for medical applications. The same test environment and datasets have been used as in section 3.1.4. The presented renderer has been implemented using the OpenGL application programming interface (API), including the OpenGL high-level shading language [SA06]. All following evaluations are based on the operation data structure (section 3.1.1). Timings have been performed for random rotations and a screen resolution of 1280×1024 pixels.

Surface Rendering

Table 3.6 shows the rendering performance in frames-per-second (FPS) for point-based isosurface visualization compared to volume raycasting and GPU-accelerated polygonal rendering. The datasets and isovalues are derived from table 3.4 and rendering is based on circular splat shapes. Note that the presented rendering technique is not restricted to isosurfaces, which have been generated from structured volume grids. It can be utilized also for general surface visualization.

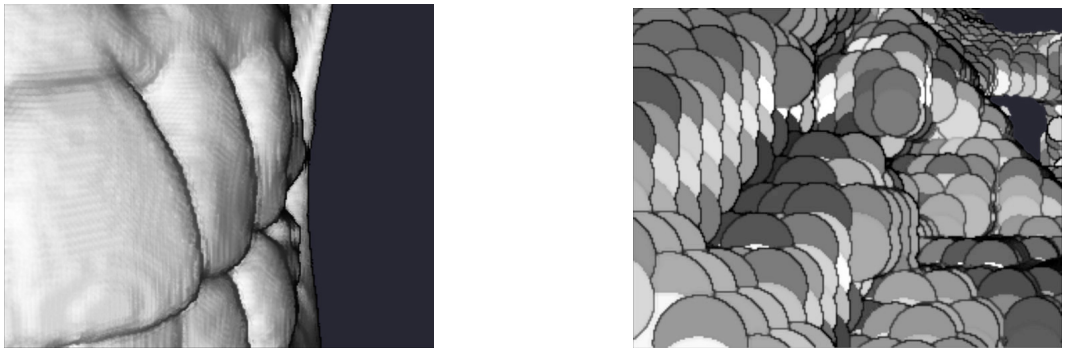
Dataset	Isovalue	Splatting	Shading	Total	Raycasting	Polygonal
Head	1077	13 ms	15 ms	36.6 FPS	0.9 FPS	170.0 FPS
	1730	13 ms	11 ms	41.7 FPS	1.0 FPS	166.7 FPS
Abdomen	807	46 ms	20 ms	15.2 FPS	0.6 FPS	6.4 FPS
	1125	45 ms	19 ms	15.8 FPS	0.5 FPS	5.3 FPS
Legs	2714	39 ms	20 ms	16.9 FPS	0.3 FPS	3.0 FPS
	3053	62 ms	20 ms	12.2 FPS	0.3 FPS	1.6 FPS
Dry Skull	1900	49 ms	16 ms	15.4 FPS	0.2 FPS	1.8 FPS
	2230	12 ms	12 ms	42.2 FPS	0.1 FPS	22.1 FPS
Visible Female	781	65 ms	12 ms	13.0 FPS	0.3 FPS	1.4 FPS
	1260	50 ms	11 ms	16.5 FPS	0.2 FPS	1.7 FPS

Table 3.6: Point-based isosurface rendering compared to traditional techniques. For raycasting and polygonal rendering, shading calculations are performed on-the-fly.

The average point rendering rate for smaller datasets (e.g. *Head* and *Dry Skull* at isovalue 2230) is between 40 and 50 millions of points per second (MPS). The renderer achieves 70 to 80 MPS for models with higher point counts. The reason for this difference is the rasterization overhead during rendering of large splats. The point-based solution is the best alternative for mid- and high-resolution objects regarding interactivity of the rendering. Higher performances could be achieved by incorporating an occlusion culling technique for point-based surface visualization [GBP04a]. Nevertheless, this requirement would be hard to realize with respect to the general rendering technique described in this work. A contradiction to the universal approach would have to be expected (one component for surface and volume data).

Point-based splatting and shading timings refer to the two steps of the *1+1* rendering approach described in section 3.2.2. In this context, normal estimation is based on arithmetic averaging of micronormals. A shading performance degradation of ca. 50 % has been observed for spherical averaging. The shading speed depends mainly on the screen resolution and the number of "empty pixels", which the reduce fragment computations. Another factor is the scaling of the viewport resolution, based on the minimal projected splat size (section 4.2.2).

The raycaster performance could be further improved by implementing empty-space-skipping and occlusion culling, e.g. using a hierarchical data structure. The input for the polygonal renderer are unoptimized triangle meshes, generated by the Marching Cubes



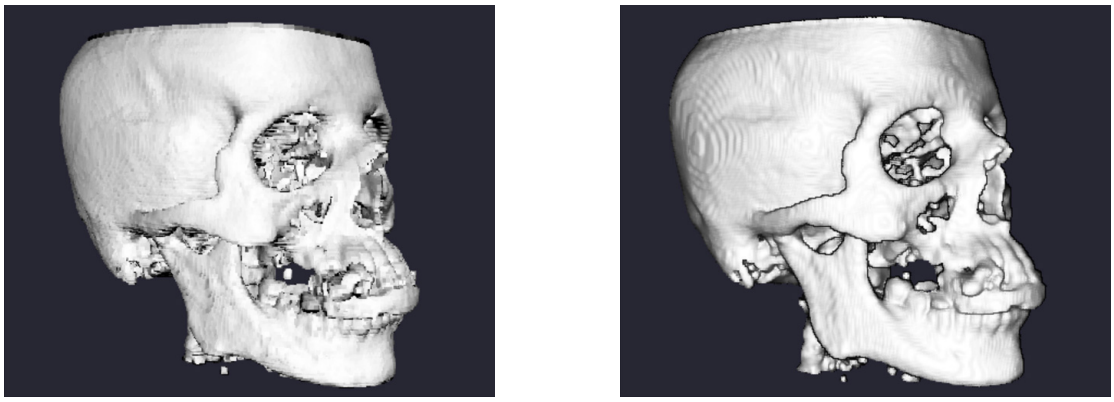
(a) Shading artifacts (b) Splat structure

Figure 3.20: Close-up views of a point-based surface model.

algorithm. The relative poor measurement results show that the associated mesh-based pipeline is dependent on further processing steps. For example, the use of a decimation technique would significantly reduce the overall rendering times (at the cost of additional preprocessing).

Figure 3.23 at the end of this section shows a collection of isosurface visualizations for selected datasets. Point-based rendering is compared to raycasting and polygonal rendering (with enabled backface culling). The visual quality for point rendering is lower than for the traditional techniques. As illustrated in figure 3.20(a), small "band" and "ring artifacts" are visible. Finer structures appear enlarged due to the spatial extents of the point primitives. There are three main factors that determine the surface visualization quality of the presented algorithm. First, generating only one point per volume cell effects in a noticeable coarse surface approximation, especially for lower dataset resolutions. A straight-forward solution to this problem is the supersampling of the volume data at the cost of an increased primitive count (figure 5.5(b)). The second source of visual artifacts is the cleft structure of the depth image, resulting from the utilization of a simple splat shape. As illustrated in figure 3.20(b), each splat is associated only with a constant depth. This results in visibility determination just at splat accuracy. Additionally, it may also lead to small "flickering" effects during rotational changes of the view point. The third parameter with a strong influence on the surface rendering quality is the normal estimation technique. Image-based techniques usually suffer from noise, image discontinuities, numerical instability and utilization of a small sample set during calculations. Figure 3.24 at the end of this section shows a comparison between the approach in this work and previous solutions.

Previous techniques are not well suited for deferred splat shading. Increasing the size of the normal estimation operator based on the projected splat size effects in an intensification of "dark band artifacts", especially for the forward and central differences as well as Magnusson's et al. Sobel filter. Gordon's et al. and Tiede's et al. techniques solve this problem satisfactorily, but at the cost of a noisy image appearance. Their handling of image discontinuities does not apply well to the irregular splat structure in the GS-



(a) Gray-level gradient shading (b) Micronormals averaging

Figure 3.21: Normal estimation in volume and image space.

buffer. Micronormals averaging minimizes the effect of dark image edges and provides the smoothest representation. Nevertheless, small visual artifacts are still noticeable. On the one hand, further improvement of the image quality could be achieved by the denoising of the 2D height-field, defined by the depth image. On the other hand, a straight-forward application of a standard smoothing filter is not an optimal solution. It would increase the effect of "dark bands" due to the amplification of discontinuities in the image [Car96]. Nevertheless, such artifacts can be interpreted as "edge enhancement" and improve the convey of complex structures [RE01].

Moreover, the presented algorithm has been tested in comparison to standard gray-level gradient shading, i.e. the computation of normals in volume space using the central difference operator [RL00]. As illustrated in figure 3.21, the quality improvement of micronormals averaging is superior. Although, gray-level gradient shading minimizes the highlighting of volume grid "jags", the resulting surface appears noisier. The main reason is the utilization of a simple splat shape in combination with constant normals for each point. Fine structures, especially in the eye socket of the *Head* dataset, are more perceptible for the image-based approach.

In general, the presented surface renderings have an adequate quality for high-resolution models. Additionally, it should be kept in mind that most medical volume datasets are noisy and complex. Therefore, a certain amount of aliasing is usually acceptable, because many artifacts are even hard to notice. The visual human perception of image quality depends on a large set of rendering and psychological parameters [YCK92]. Therefore, uncomplicated rendering and normal estimation techniques must not automatically imply the inferiority of such solutions.

Volume Rendering

Table 3.7 gives an overview of additional datasets that will be used in the remainder of this section. The *Ventricles* and *Vessels* datasets have 16 scalar bits allocated and 12 bits

used. The *Heart* has eight scalar bits. *Ventricles* and *Heart* are MRI datasets, *Vessels* is an angiography scan. For information about the originators of the volumetric datasets refer to the acknowledgments at the beginning of this work.

Dataset	Original resolution	Memory	Isotropic resolution	Memory
Ventricles	$348 \times 348 \times 97$	22 MB	$348 \times 348 \times 139$	32 MB
Heart	$352 \times 352 \times 256$	30 MB	$352 \times 352 \times 256$	30 MB
Vessels	$512 \times 512 \times 512$	256 MB	$512 \times 512 \times 512$	256 MB

Table 3.7: Volume datasets used for point-based rendering.

Table 3.8 includes the timings for point-based MIP rendering of full volumetric datasets, i.e. using the complete scalar range. During preprocessing, points have been placed at the original volume grid samples in order to avoid interpolation of scalar values. The results are compared to those of the traditional direct volume renderers, including software raycasting and graphics-hardware-accelerated slicing via 3D texture mapping.

Dataset	Splatting	Shading	Total	Raycasting	3D Texture
Head	782 ms	5 ms	1.3 FPS	0.9 FPS	32.7 FPS
Abdomen	2671 ms	4 ms	0.4 FPS	0.5 FPS	9.2 FPS
Vessels	2377 ms	4 ms	0.4 FPS	0.6 FPS	1.1 FPS

Table 3.8: Point-based full MIP rendering compared to traditional techniques.

The *Head* and *Abdomen* datasets have been introduced in section 3.1.4. The point-based renderer requires the isotropic versions of the input volumes. The raycaster and 3D texture-mapping renderer are capable of processing the original resolutions. Both have not been optimized and further improvements, like empty-space-skipping and bricking, would effect in better performance. The average rendering rate for point-based visualization is approximately 64 MPS. Compared to surface rendering, the shading step is 2-4 times faster, because no normal estimation has to be performed. Nevertheless, the visualization of a full point-based volume is less efficient compared to the traditional techniques and the best performance is achieved by the 3D texture-mapping renderer. The main reason is the per-vertex overhead of the point-based pipeline. Each element is considered as an own primitive, requiring individual transformation and projection. Additionally, for each splat a transfer function look-up is required. Reduced memory traffic and extra speed-up could be achieved by splatting of material values in the GS-buffer. Then, the look-up could be executed during the shading step in image space. Nevertheless, such an approach would require a global material database for all objects in the scene. The realization of this idea would be a hard challenge regarding the high scalar bit depths and variations of medical datasets.

Solutions for providing interactive volume visualization with points will be presented in the remainder of this work. Finally, figure 3.25 at the end of this section shows a visual comparison between the discussed techniques. A MIP rendering provides an informative

overview of the whole dataset. All techniques achieve comparable visual quality. The traditional techniques exhibit slightly more sharp results.

Extended Surface Rendering

The aforementioned volume renderings have been generated for the full scalar range. This is not an optimal scenario, because the flexibility feature of point-based representations is not exploited. Better performance and memory results can be achieved by irregular volume sampling and discarding of irrelevant elements during preprocessing, resulting in extended surfaces (section 3.1). Table 3.9 includes the corresponding measurements using MIP as well as surface and hybrid rendering with material information.

Dataset	Scalar range	Points	Type	Splatting	Shading	Total
Vessels	575, 4095	1,551,100	MIP	28 ms	5 ms	30.7 FPS
Abdomen	1125, 4095	5,937,926	MIP	98 ms	5 ms	9.7 FPS
Heart	100, 255	6,440,014	Surface	103 ms	19 ms	8.2 FPS
Ventricles	186, 4095	7,778,815	Hybrid	208 ms	17 ms	4.4 FPS
Head	1077, 4095	8,637,084	MIP	238 ms	5 ms	4.1 FPS
Dry Skull	1900, 4095	8,974,213	Surface	142 ms	29 ms	5.9 FPS

Table 3.9: Point-based rendering of extended surfaces.

Given a reduced scalar range, all datasets can be visualized with significantly higher frame rates compared to full volume rendering. The visualizations of the datasets *Head*, *Abdomen* and *Vessels* achieve speed-up factors of 3-72 with respect to table 3.8. An average point rate of 50-60 MPS is reached. Lower resolution datasets (*Head*) and hybrid visualization types (*Ventricles*) exhibit a reduced point rate of approximately 37 MPS. The reason is the rasterization overhead for large splats in combination with additional computations due to the transfer function look-ups.

Point-based extended surfaces are a promising compromise between full volumetric and surface visualizations. In the context of MIP rendering, representations with extended surfaces make it possible to reduce the scalar range and free memory for irrelevant portions of the data. This results in faster visualization, whereas important information can be preserved by applying conservative scalar range estimations during a priori exploration. An extended surface MIP rendering of the dataset *Head* is presented in figure 3.26(c). Its full volume counterpart can be seen in figure 3.25(a). Extended surfaces add volumetric information and fill the hollow spaces that are available in pure surface representations. Combining such visualizations with clip planes provides a similar effect as the traditional overlay planes, available in many medical applications (figure 1.6(a)). A traditional implementation typically requires the access to the original volume, a triangle-mesh and textured planes (figure 1.7). The presented point-based solution requires only a single software component. Furthermore, it provides more configurability for the user by making it possible to define the "thickness" of the surface, for example. In figure 3.26 at the end of the chapter, various extended surface renderings are presented.

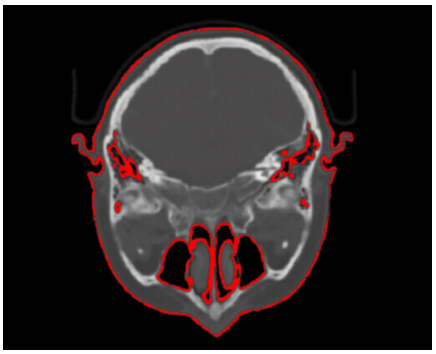
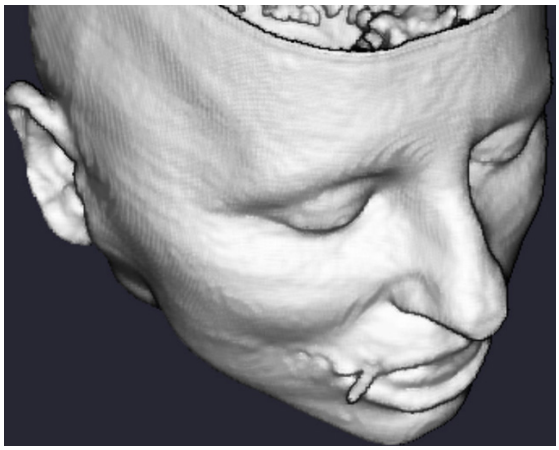


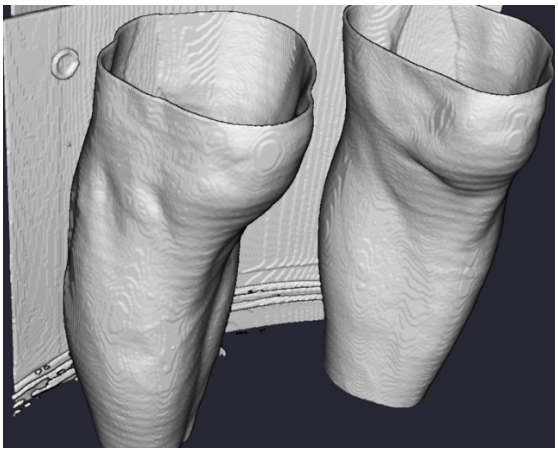
Figure 3.22: Point-based slice view with surface contour (red color).

Furthermore, clip planes are a useful tool for visualization of extended surfaces. For MIP it is possible to interactively cut away structures that have a negative impact on the rendered image, e.g. bone structures that lie in the background of the scene (figures 3.26(e) and 3.26(f)). Additionally, "fat clip planes" make it possible to simulate traditional 2D slice and polygonal contour views (figure 3.22). A similar visualization, created with ordinary volume- and polygonal slices, is illustrated in figure 1.6(b). The point-based contour shows a noticeable thickness, because each point has its own spatial extent. In contrast, polygonal contours are drawn using thin lines, resulting from plane/triangle intersections. Therefore, their visual appearance is sharper.

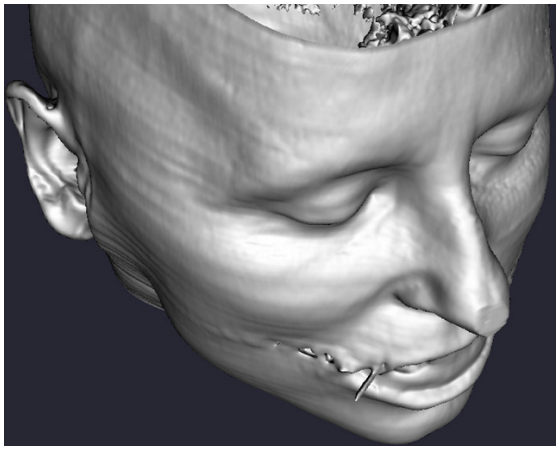
The reported rendering performances, especially in tables 3.8 and 3.9, are not satisfactorily for high-resolution datasets. For an experience of quasi-smooth interactivity, frame rates of at least 15 FPS are required. A solution to this problem is progressive refinement and level-of-detail rendering. These techniques, including adaptations to the presented point-based visualization pipeline, will be described in the next chapter.



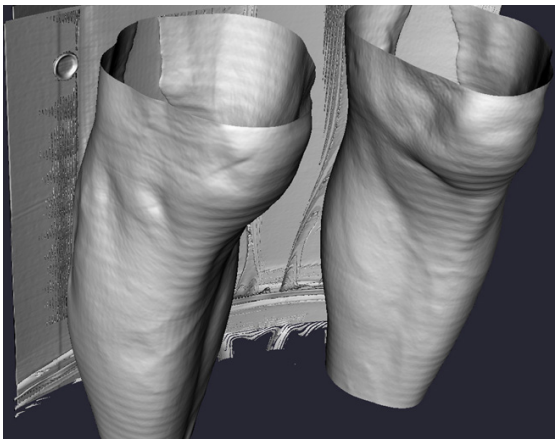
(a) Point-based



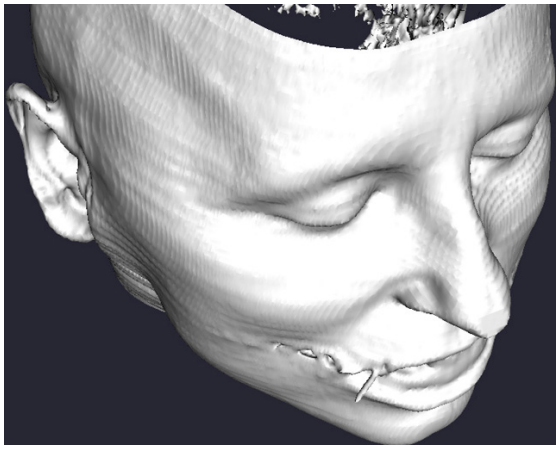
(b) Point-based



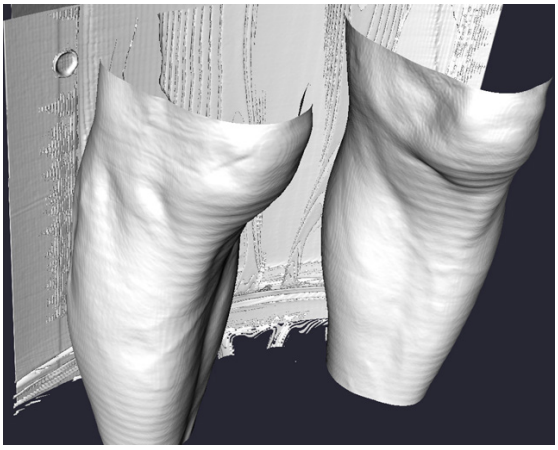
(c) Raycasting



(d) Raycasting

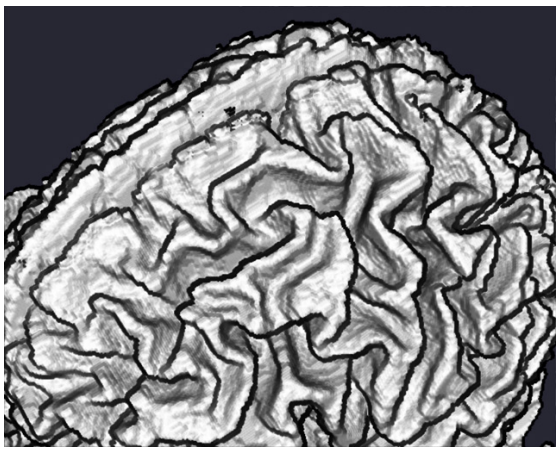


(e) Polygonal

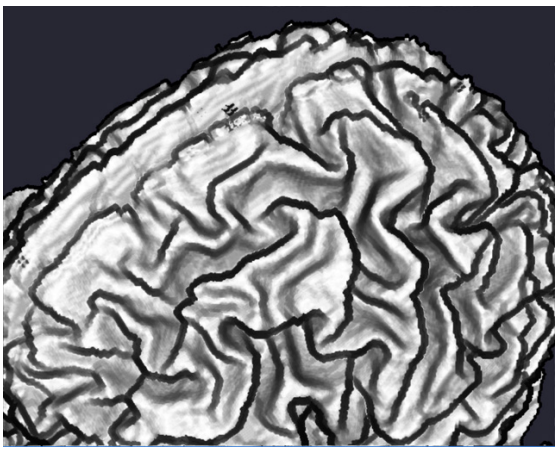


(f) Polygonal

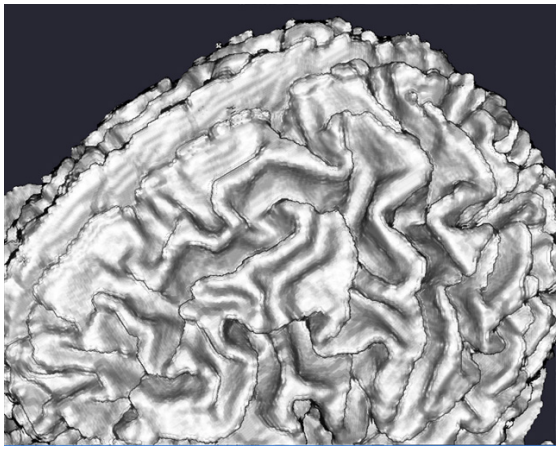
Figure 3.23: Isosurface visualizations of *Head* (isovalue 1077) and *Legs* (isovalue 2714).



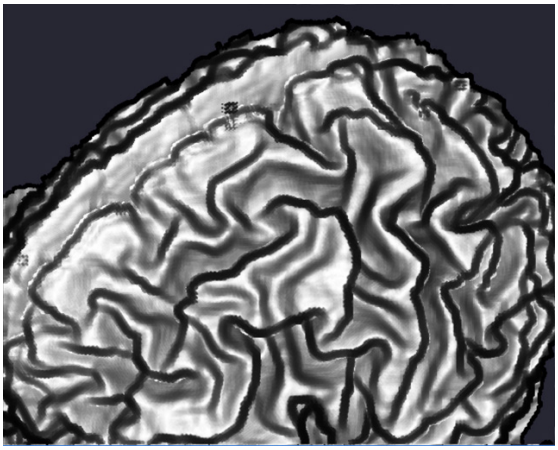
(a) Forward difference



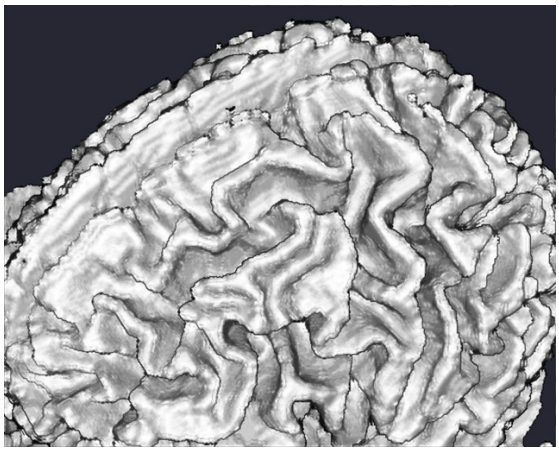
(b) Central difference



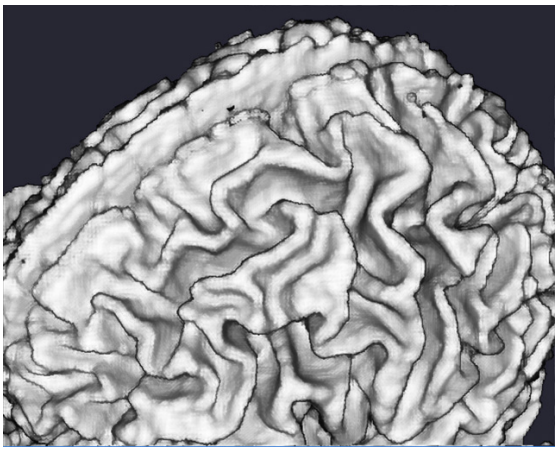
(c) Gordon et al. [GR85]



(d) Sobel filter [MLD88]

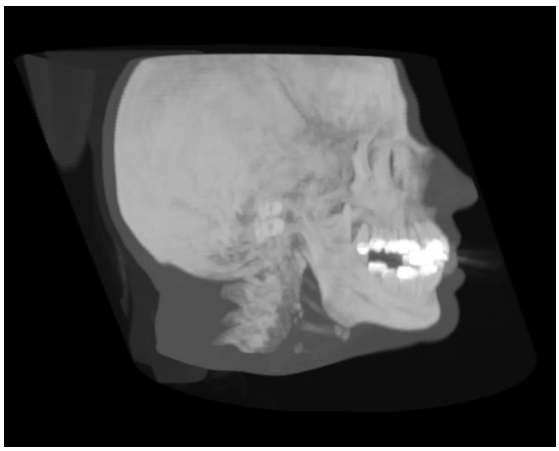


(e) Tiede et al. [THB+90]

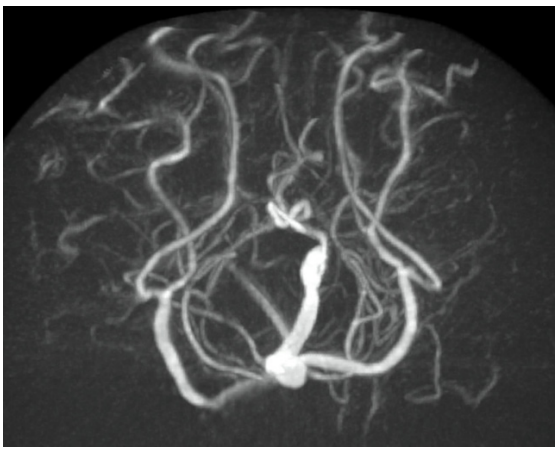


(f) Micronormals averaging

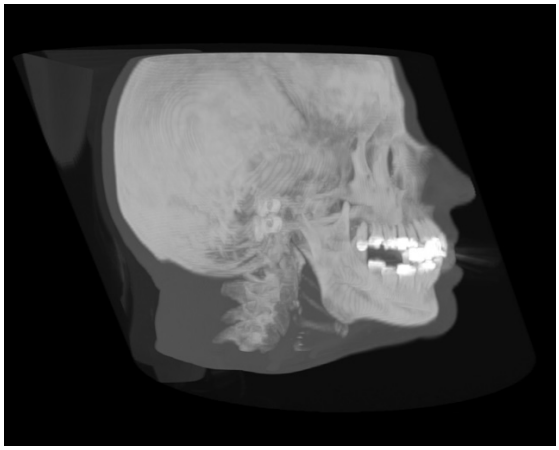
Figure 3.24: Image-based normal estimation techniques for a MRI brain dataset.



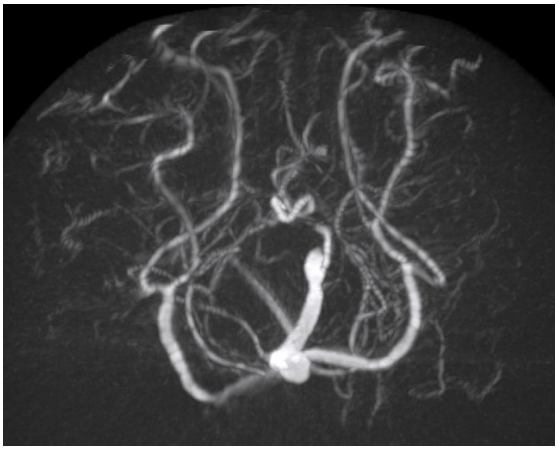
(a) Point-based



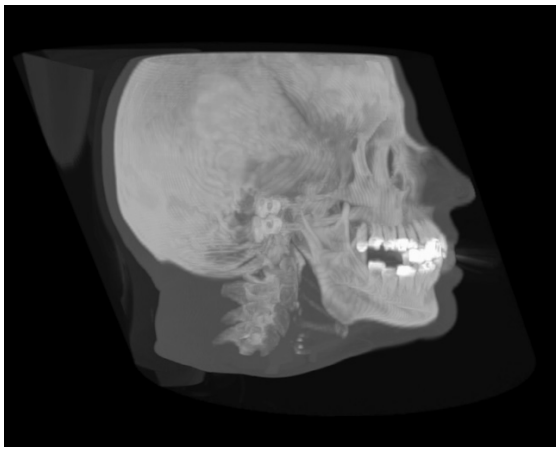
(b) Point-based



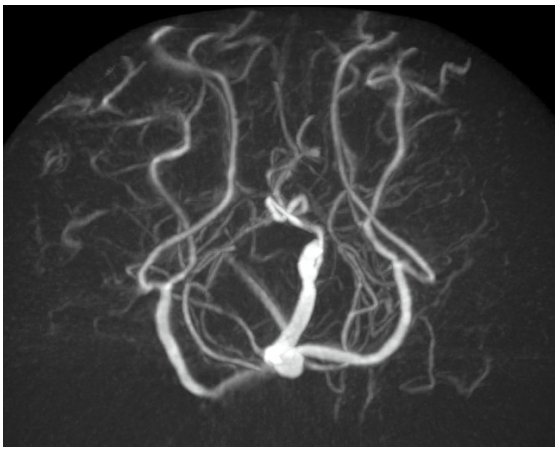
(c) Raycasting



(d) Raycasting

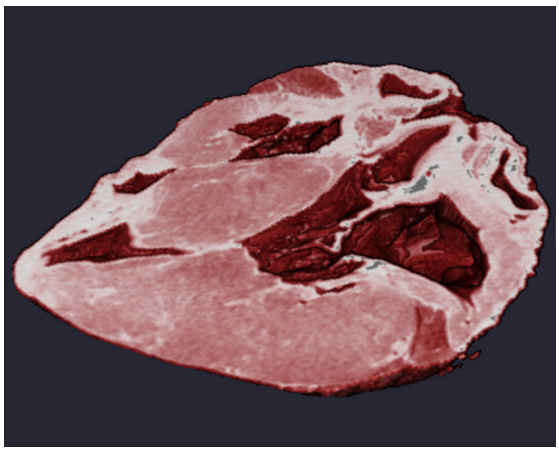


(e) 3D texture-mapping

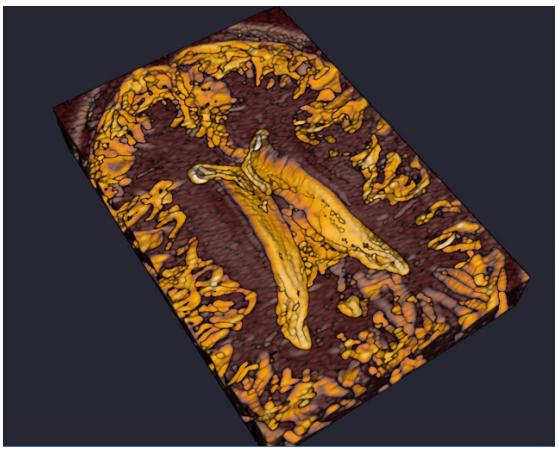


(f) 3D texture-mapping

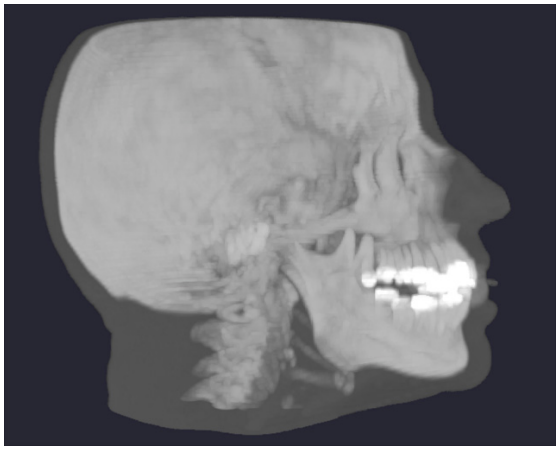
Figure 3.25: Full MIP volume rendering of *Head* and *Vessels*.



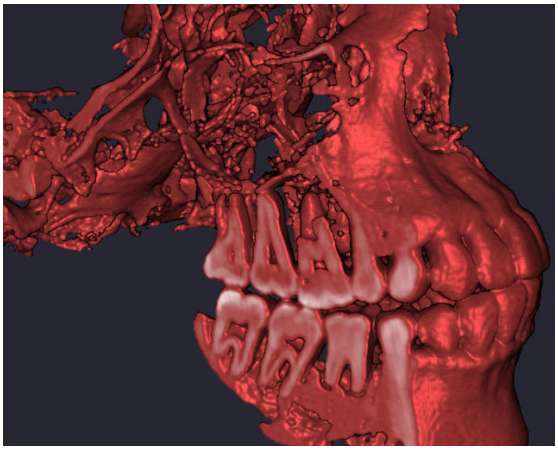
(a) Surface visualization of *Heart*



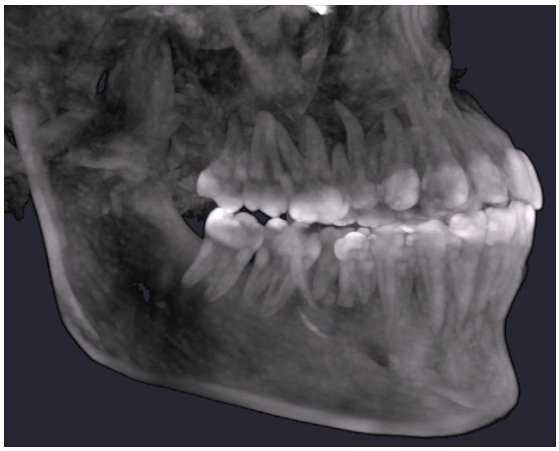
(b) Hybrid visualization of *Ventricles*



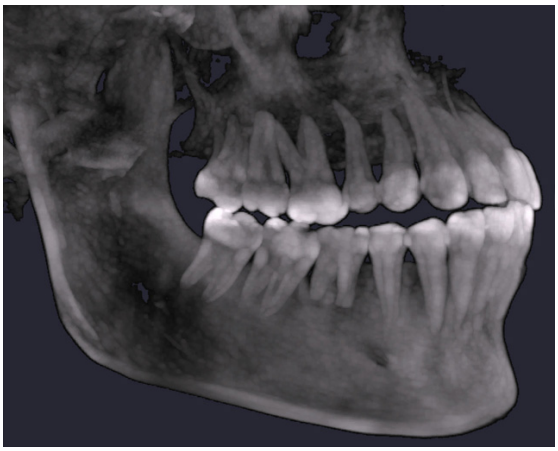
(c) MIP visualization of *Head*



(d) Surface visualization of *Dry Skull*



(e) MIP visualization of *Dry Skull*



(f) MIP visualization with clip plane

Figure 3.26: Renderings of extended surfaces using transfer functions and clip planes.

Chapter 4

Adaptive Display of Point Data

In this chapter solutions for preserving interactivity during the visualization of complex point-based models will be presented. The first part is a progressive refinement framework that has been adapted to typical medical scenes. Moreover, it includes a sequential level-of-detail algorithm that makes it possible to access coarse geometric representations very efficiently. Additionally, it will be shown how to integrate all these techniques into the point-based visualization pipeline described in chapter 3.

Perhaps the most striking challenge for 3D visualization applications in the future will be the development of interactive rendering techniques in spite of continuously growing datasets. High frame rates are important to achieve an illusion of real-time exploration of the virtual scene, especially in the medical context. One possible solution is the utilization of so-called *adaptive display algorithms* that adjust the image quality in order to guarantee interactivity [FS93]. The focus in the following sections will be on this class of techniques for point-based models.

A straight-forward solution would be to barter performance for model complexity during preprocessing, e.g. by resampling the dataset to a lower resolution. Unfortunately, such a static simplification effects usually in the loss of important details, apparent primarily for close-up views. Therefore, the rendering component should have access to the original high-resolution dataset and control the geometric complexity and frame rate on-the-fly. The *level-of-detail* (LOD) is usually adapted dynamically based on a screen space error metric by traversing only a subset of available geometric primitives, generating a "smaller" model with less details, or by utilizing algorithms from image-based rendering. The main idea of the latter approach is to replace "less important" subsets of the original model with images or depth textures [AL99]. Nevertheless, many pure LOD systems usually cannot guarantee stable frame rates or at least frame rate bounds [FS93]. For an overview of existing LOD techniques for point-based rendering please refer to section 2.2.

A possibility for visualization with respect to a target frame rate is *progressive refinement*. Instead of processing the complete scene at once, rendering is performed in background or split-up into multiple passes. As long as no interaction occurs, the rendering is refined in time. On the one hand, the user is confronted in between only with an

approximated output image with visual errors. On the other hand, control is returned to the application, improving the overall impression of interactivity. Therefore, progressive refinement handles the classic trade-off between rendering speed and image quality.

One important goal during the design of the adaptive display solution in this work has been the exploitation of the flexibility of point data. The associated progressive refinement framework is combined with a sequential LOD technique that makes it possible to control simply the geometric complexity by the number of points. Various priority and point density functions can be integrated for adaptive visualization of different object parts. All described techniques are universal, i.e. work for surface and volumetric models. Moreover, they provide reasonable results even after modification of the underlying dataset (e.g. after cutting of an object), without additional postprocessing.

4.1 Progressive Refinement Framework

Progressive refinement techniques have a long history in streamed image transmission, polygonal rendering, raycasting and volume splatting [WGH83, BFGS86, Lev90b, LH91]. Bergman's et al. 20 years old requirements for the development of a smart progressive refinement system are still valid today and have been considered during the design of the point-based framework in this work:

- Concentrate on regions of a dataset where they make the most visual difference
- Visualize intermediate results and interrupt refinement on every user input
- Minimize processing costs by reusing intermediate results from previous frames

The preservation of a uniform or at least bound frame rate during exploration of polygonal scenes was investigated in detail by Funkhouser et al. in 1993 [FS93]. The authors have tried to find an optimal configuration of so-called *object tuples* - each associated with a scene object, hierarchical detail level and rendering algorithm - in order to maintain a target frame rate. They argued that the main challenge is the prevention of "overshoots" and "oscillations" that are typical for *reactive* algorithms, which select the detail level based on the previous render time. Therefore, Funkhouser et al. introduced a *predictive* approach, including a benefit and cost function for each entity in the scene, and used it to estimate the optimal object tuple set before rendering. A reliable definition of both functions can be a hard task for polygonal and complex CAD scenes. It will be shown that utilizing a point-based approach makes it possible to implement adaptive display algorithms for typical medical scenes in a much simpler way.

Basic Approach

A popular and simple progressive refinement scheme is the so-called "rendering on a time budget". It can be described by the following three events:

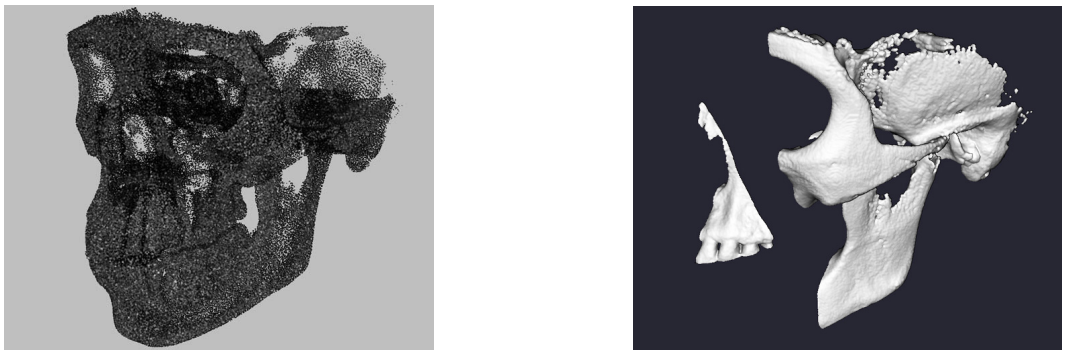
- **Timeout -** Break rendering, when the render time exceeds a threshold.
- **Idling -** Continue rendering, when the application gets idle again.
- **Interaction -** On each user input stop refinement and restart rendering.

The timeout is usually specified by a user-defined target render time. There are two different possibilities to present visualization results to the user. First, the display is updated after each timeout and when the rendering is completed. Such an approach makes it possible to reuse the information from previous refinement passes e.g. by preserving the back- and z-buffer contents. The user can pursue the update of the scene on-the-fly. In case of splatting, points have to be rendered at the original point size, so intermediate images may contain holes (figure 4.1(a)). Another possibility is to update the display only after the first and last refinement pass. In such a case, the user is only confronted with the coarse and the final images. This approach makes it also possible to close the holes in the coarse image via rendering with adaptive point sizes, as will be described in section 4.2.2.

The simple timeout strategy of the basic refinement algorithm has a significant limitation. During interaction only a subset of the complete scene is visualized, effecting in a limited view for the user. As illustrated in figure 4.1(b), important object parts could be missed, making the orientation difficult. As will be described in the remainder of this section, the problem is solved by the adaptive display technique.

Adaptive Approach

In contrast to the basic algorithm, the adaptive renderer considers all points in the scene on each refinement pass. The point density is adapted during traversal in order to guarantee a desired frame rate. Regions that are considered as more "important" are visualized with a higher point density. Such an approach can be considered as an implicit "focus-plus-context" visualization technique, based on a "variable resolution LOD" [LW90]. Another interpretation is "importance sampling" for point-based LOD rendering.



(a) Holes in point model (b) Missing object parts

Figure 4.1: Problems with simple timeout-based progressive refinement.

The concept of adaptive display has been utilized previously in the context of traditional data structures and alternative point-based rendering techniques, including variable LOD visualization for polygonal meshes and importance sampling for point-based Monte Carlo rendering [RLB+01, CSK03]. Another example is importance sampling for ray tracing. The main idea is to concentrate more ray samples in regions that contribute most to the rendered image [DH92]. Applied to volume ray casting, it results in a technique called *β-acceleration*, i.e. the sampling density is decreased with the depth along the ray. Another area of application is slice-based volume rendering with 3D textures [LHJ99, WWH+00]. The input volume is converted into a multiresolution brick hierarchy, like the octree. The resolution level of each volume brick is derived from the distance to a "focus point" (e.g. based on the camera location). Finally, Meng et al. have developed a method for streaming transmission of point data with gaze-guided control of the detail level [MZ03].

Overview:

In the following, an overview of the adaptive display approach will be provided based on the notation of Funkhouser et al. [FS93]. Assume that the scene consists of several point objects, each built from a set of point buckets. Each bucket is implemented with an array data structure for efficient sequential processing. The best visual results can be achieved for data structures that consist of large buckets with a high geometric cohesion. An example is the operation data structure, described in section 3.1.1. For other cases, like the exploration data structure, it is recommended to utilize the basic approach for progressive refinement. On each refinement pass, the frame rate is controlled using the following two-level approach:

Object level - In the first level, each object O_i with $1 \leq i \leq M$ is associated with a fraction of the user-supplied rendering time t_{target}. Let N_i be the total number of points for object O_i and $w_{RT} \in [0, 1]$ the *render time weight*. Then, the maximal render time t_i for O_i is defined as follows:

$$t_i = t_{target} \cdot \left(\frac{N_i \cdot w_{RT}}{\sum_{k=1}^{M} N_k} + \frac{1 - w_{RT}}{M} \right) \tag{4.1}$$

The left summand corresponds to the calculation of the render time based on the fraction of points in the current object. The right summand refers to the number of objects in the scene. Setting higher values for w_{RT} will favor complex objects, which consist of many point primitives. Lower values effect in a more "fair" distribution of the render time, useful for increasing the "importance" of smaller objects, like surgical tools. It has been observed that $w_{RT} = \frac{1}{4}$ produces reasonable results in practice.

Bucket level - Based on the concepts of Funkhouser's et al. work, the control of the progressive refinement for each point object is considered as the following optimiza-

tion problem: Let $B_{i,j}$ with $1 \leq j \leq M_i$ be the j-th bucket for point object O_i and $N'_{i,j} \leq N_{i,j}$ the actual number of points used for visualization of $B_{i,j}$. Then:

$$\text{Objective:} \quad \sum_{k=1}^{M_i} \mathit{Benefit}(B_{i,k}, N'_{i,k}) \rightarrow max! \tag{4.2}$$

$$\text{Constraint:} \quad \sum_{k=1}^{M_i} \mathit{Cost}(B_{i,k}, N'_{i,k}) \leq t_i \tag{4.3}$$

The *Benefit* function provides the "visual value" of a bucket $B_{i,j}$ for the current view into the virtual scene, if rendered with the point count $N'_{i,k}$. It can be interpreted as the "importance" of the corresponding bucket. The *Cost* function returns a prediction of the rendering time for a bucket with respect to the associated point count. Funkhouser's et al. have used a greedy approximation algorithm to solve the aforementioned optimization problem. In this work, the following two-step approach is applied:

1. In the first step, all buckets are sorted inversely based on $\mathit{Benefit}(B_{i,k}, N'_{i,k})$ with respect to the ceteris paribus assumption $N'_{i,k} = N_{i,k}$, i.e. each bucket is considered with its full point count. The resulting order reflects the *refinement priority* for rendering and will be addressed in section 4.1.1.
2. In the second step, the actual point count $N'_{i,k}$ is adapted with respect to the satisfaction of the optimization constraint. This procedure corresponds to the control of the point density for each bucket, as will be described in section 4.1.2.

The presented algorithm does not guarantee globally optimal solutions, but it produces reasonable results for point-based rendering. Moreover, it can be implemented very efficiently.

Estimating the render time:

A reliable prediction of the render time per point bucket is fundamental for a stable frame rate control. Wimmer's et al. analysis has shown that it is hard to estimate for current graphics boards due to the high parallelization of the hardware units and missing timing functionality support in most rendering APIs [WW03]. In this work, a simple solution is used, which is not perfect, but produces reasonable results in practice.

First, the *point render rate* R_i per object O_i is estimated, i.e. the number of points that can be rendered per time interval. In order to adapt smoothly to changes in the scene, an on-the-fly weighting of the render rate with the previously estimated value is performed. Time measurements are executed via CPU and GPU synchronization for each point object. Finally, the following cost heuristic is applied for each point bucket:

$$Cost(B_{i,k}, N'_{i,k}) = \frac{N'_{i,k}}{R_i} \tag{4.4}$$

This purely point-oriented approach does not consider the pixel rasterization overhead during rendering of large splats. A solution to this problem will be discussed in section 4.2.2.

4.1.1 Refinement Priority Estimation

The computation of the refinement priority consists of the view-dependent estimation of the point bucket benefits and sorting of buckets in decreasing order. This procedure has some conceptual similarities to Kim's et al. view-dependent streaming of progressive meshes [KLK04]. Their server-to-client transmission order is determined according to the visual importance with respect to the client's view point.

The implementation in this work is based on the abstract iterator interface for point data structures, as described in section 3.2. Therefore, it can be utilized directly with the aforementioned renderer component. In the following, two examples will be provided for the estimation of the refinement priority.

Depth-Based Iteration

During exploration of a typical medical scene, all objects that are located in front of the viewer are usually considered as the most important ones. Therefore, the simplest benefit function for a point bucket can be defined as follows: Let $\vec{b}_{i,j}$ be the center point for $N_{i,j}$ points in bucket $B_{i,j}$, $\vec{c}$ the current camera position and $D : \Re^3 \times \Re^3 \rightarrow \Re$ the Euclidean distance function. Then:

$$Benefit_{Depth}(B_{i,k}, N_{i,j}) = \begin{cases} D(\vec{b}_{i,j}, \vec{c})^{-1}, & B_{i,j} \textit{ is visible} \\ 0, & \textit{Otherwise} \end{cases} \tag{4.5}$$

For the 3D point grid data structure, the resulting front-to-back iteration is implemented using three nested loops, each traversing a different grid axis. All depth-based benefit values can be calculated efficiently in $O(M_i)$ time. As illustrated in figure 4.3(a), the resulting visual quality may be not adequate for objects with complex structures.

Visibility-Based Iteration

A more sophisticated metric for the visual benefit should consider the occlusion between different point buckets. Therefore, it is suggested to associate the benefit with a *bucket visibility* value.

The visibility determination problem is interpreted as *cone tracing* of a sphere-based scene [Ama84]. Each point bucket is approximated with a bounding sphere and the included number of points is considered as its "opaqueness". All spheres generate the so-called *occlusion cones* based on the current camera position (figure 4.2). The overlap

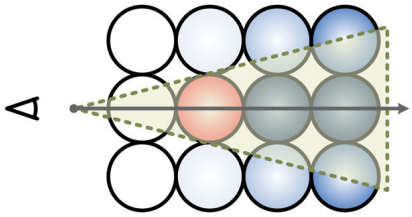
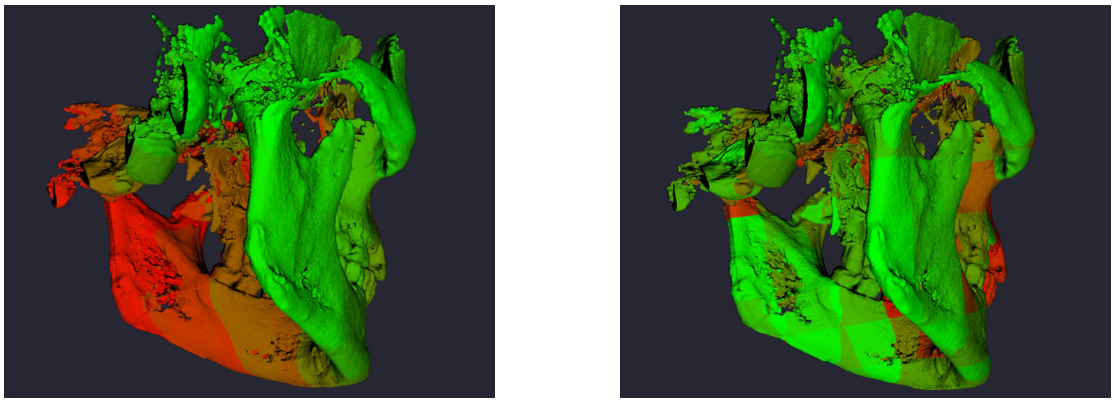


Figure 4.2: 2D illustration of an occlusion cone. The red circle represents the occluder. The remaining circles are colored from light blue (high) to dark blue (low) based on their visibility from the current viewing position.

between a bounding sphere and all intersecting occlusion cones is used to derive the visibility of the corresponding point bucket as follows: Let $V_{i,j}^S$ and $V_{i,j}^C$ be the bounding sphere and the visibility cone volumes, respectively, for $N_{i,j}$ points in bucket $B_{i,j}$. Then, the benefit is estimated by a weighted summation of the numbers of points, which potentially occlude $B_{i,j}$:

$$Benefit_{Visibility}(B_{i,j}, N_{i,j}) = \left(\sum_{k=1,k\neq j}^{M_i} \frac{|V_{i,j}^S \cap V_{i,k}^C|}{|V_{i,j}^S|} \cdot N_{i,k} \right)^{-1} \tag{4.6}$$

The intersection volume can be approximated using the minimal distance between the cone center ray and the sphere center. A straight-forward computation of all benefit values has the time complexity $O(M_i^2)$. An illustration for visibility-based refinement is shown in figure 4.3(b).



(a) Depth-based (b) Visibility-based

Figure 4.3: Refinement order priorities, colored from green (high) to red (low).

4.1.2 Point Density Control

The visualization of a point object O_i is based on an ordered and bucket-wise traversal of the included point elements. The point density is adapted for each bucket using the *point density function* $\rho_i : \Re \rightarrow \Re$. It maps from an abstract *rendering progress* value to the actual point density. Let $0 \leq \rho_{min} \leq \rho_{max} \leq 1$ be the predefined minimum and maximum point densities, respectively. The definition of ρ_i is user-defined with respect to the following conditions:

- $\rho_i(0) = \rho_{max}$
- $\rho_i(1) = \rho_{min}$
- ρ_i is monotonic decreasing
- ρ_i must satisfy cost constraint 4.3

The last point will be detailed in the remainder of this section. Additionally, examples for point density functions will be presented, including a high-level algorithm for adaptive display.

Satisfying the Cost Constraint

The actual number of points $N'_{i,j}$ for array-based rendering of bucket $B_{i,j}$ can be estimated with the point density function ρ_i. First, the sorted sequence of buckets by the refinement priority is interpreted as a sequential list of N_i point elements, representing the entire point object. The mapping between the global point index $n_i \in \{1, \ldots, N_i\}$ and the abstract progress value $\varphi_i \in [0, 1]$ is accomplished as follows:

$$\varphi_i = \frac{n_i - 1}{N_i - 1} \tag{4.7}$$

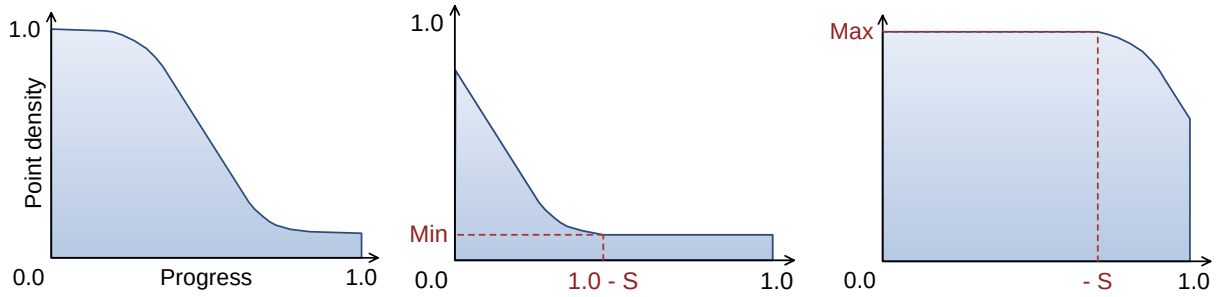
Let $\varphi^-_{i,j}$ and $\varphi^+_{i,j}$ be the progress values corresponding to the first and last point element of the j-th bucket, respectively. Then, the integral value of the point density function is interpreted as the fraction of the point count, i.e.:

$$N'_{i,j} = N_{i,j} \cdot \int_{\varphi^-_{i,j}}^{\varphi^+_{i,j}} \rho_i(\varphi)\ d\varphi \tag{4.8}$$

Using the cost function 4.4 and equation 4.8, the cost constraint 4.3 is rewritten as:

$$\sum_{k=1}^{M_i} \frac{N_{i,k} \cdot \int_{\varphi^-_{i,k}}^{\varphi^+_{i,k}} \rho_i(\varphi)\ d\varphi}{R_i} \leq t_i \tag{4.9}$$

Finally, by considering the sum of per-bucket point counts and integral contributions for the whole object, it is:



(a) $S = 0$ (b) $S > 0$ (c) $S < 0$

Figure 4.4: Point density functions for different values of the shift parameter.

$$\int_0^1 \rho_i(\varphi)\ d\varphi \leq t_i \cdot \frac{R_i}{N_i} \tag{4.10}$$

Example:

Relation 4.10 can be used to derive point density functions as follows: Assume that each ρ_i is constructed using a *pattern function* $\rho_i' : \Re \to \Re$ with two unknown variables $a, b \in \Re$. Examples are polynomial functions $\rho_i'(\varphi) = a + b \cdot \varphi^c$ and exponential functions $\rho_i'(\varphi) = a \cdot e^{b \cdot \varphi^c}$, for a user-defined parameter $c \in \Re$. By introducing a *density shift* $S \in [-1, 1]$, the point density function ρ_i can be expressed as $\rho_i^S : \Re \to \Re$:

$$\rho_i^S(\varphi) = \begin{cases} \rho'_{max}, & \varphi + S < 0 \\ \rho'_{min}, & \varphi + S > 1 \\ \rho_i'(\varphi + S), & \textit{Otherwise} \end{cases} \tag{4.11}$$

The shift parameter is used to control the point density with respect to constraint 4.10, i.e. increasing S effects in a decrease of the point density integral and vice versa. As illustrated in figure 4.4, this effect can be interpreted as "moving" ρ_i^S to the left and right, respectively. The unknown variables a, b and S are calculated as follows:

1. Calculate a and b using ρ_i' and the conditions $\rho_i'(0) = \rho_{max}$ and $\rho_i'(1) = \rho_{min}$. Then, set up the pattern function integral $P_i'(\varphi_1, \varphi_2) = \int_{\varphi_1}^{\varphi_2} \rho_i'(\varphi)\ d\varphi$.

2. Set up the point density integral $P_i^S(\varphi_1, \varphi_2) = \int_{\varphi_1}^{\varphi_2} \rho_i^S(\varphi)\ d\varphi$ based on equation 4.11. P_i^S can be expressed using P_i' as follows:

$$P_i^S(\varphi_1,\varphi_2) = \begin{cases} (\varphi_2-\varphi_1)\cdot\varphi_{max}, & S<0 \ \wedge \ \varphi_1,\varphi_2 \le -S \\ (-S-\varphi_1)\cdot\varphi_{max} + P_i'(0,\varphi_2+S), & S<0 \ \wedge \ \varphi_1 \le -S < \varphi_2 \\ (\varphi_2-1+S)\cdot\varphi_{min} + P_i'(\varphi_1+S,1), & S>0 \ \wedge \ \varphi_1 \le 1-S < \varphi_2 \\ (\varphi_2-\varphi_1)\cdot\varphi_{min}, & S>0 \ \wedge \ \varphi_1,\varphi_2 > 1-S \\ P_i'(\varphi_1+S,\varphi_2+S), & \textit{Otherwise} \end{cases} \tag{4.12}$$

3. Calculate the density shift S using P_i^S and condition 4.10, i.e. $P_i^S(0,1) = t_i \cdot \frac{R_i}{N_i}$: Let be $P_i^{S-} = -S\cdot\varphi_{max} + P_i'(0,1+S)$ and $P_i^{S+} = S\cdot\varphi_{min} + P_i'(S,1)$. For the integral over $[0,1]$, equation 4.12 simplifies to:

$$P_i^S(0,1) = \begin{cases} P_i^{S-}, & S<0 \\ P_i^{S+}, & \textit{Otherwise} \end{cases} \tag{4.13}$$

It is required to choose between P_i^{S-} and P_i^{S+} for the following calculations: The pattern function integral P_i' corresponds to the unshifted point density function integral P_i^S. If $P_i'(0,1)$ is smaller than the target cost value $t_i \cdot \frac{R_i}{N_i}$ of equation 4.10, then the point density function has to be shifted "to the right" in order to increase its integral value (figure 4.4). In other words, it must be $S<0$. Otherwise, it has to be shifted "to the left", i.e. S must be greater than zero.

Altogether, S can be solved as follows: If $P_i'(0,1) \le t_i \cdot \frac{R_i}{N_i}$, then use condition $P_i^{S-} = t_i \cdot \frac{R_i}{N_i}$. Otherwise, use $P_i^{S+} = t_i \cdot \frac{R_i}{N_i}$. If there is no valid solution, then φ_{min} or φ_{max} has been chosen too large or too small, respectively. For complex pattern functions it may be more practical to solve the equations numerically. For example, the monotony condition of the point density function makes it possible to perform an efficient *binary searc*h for S in the interval [-1,1].

Adaptive Rendering Algorithm

All aforementioned calculations are summarized in the high-level rendering algorithm 4 for adaptive display of point data. It can be considered as an extension of the splatting step in algorithm 3 (section 3.2). All operations have to be executed for each refinement pass.

The calculations of all subsequent rendering passes are performed on the remaining number of primitives, i.e. the $N_{i,j}$'s are getting smaller with each iteration step. The splatting of the remaining points per bucket is executed as a sequential operation on the associated point arrays. Intermediate visual results can be presented to the viewer after each pass by copying the GS-buffer content to the frontbuffer and executing shading calculations (section 3.2.2). Notice that in case of visualization with adaptive point sizes, intermediate results can only be presented after the first pass, because the GS-buffer cannot be reused by subsequent refinement steps (section 4.2.2).

Algorithm 4: Splatting step based on refinement order and point density control.

```
foreach point object O_i do
    Compute point density function ρ_i using cost constraint 4.10;
    foreach point bucket B_{i,j} ∈ O_i in reversed order of Benefit(B_{i,j}, N_{i,j}) do
        Compute progress values φ⁻_{i,j} and φ⁺_{i,j} using equation 4.7;
        Compute render point count N'_{i,j} using equation 4.8;
        Splat the resulting number of points as illustrated in algorithm 3;
    end
end
```

4.1.3 Smooth Refinement during Interaction

Interaction with the virtual point model during progressive refinement may lead to temporal artifacts, resulting in undesirable effects known as "flickering" or "popping" (figure 4.5). Such effects do not relate exclusively to the basic approach described in section 4.1. Similar artifacts may occur also for the adaptive algorithm in this work, requiring the incorporation of techniques for smooth refinement. In the remainder of this section an overview of the three main reasons for "flickering" will be provided, including suggestions for its solution:

Inaccurate time measurements - The point render rate R_i in equation 4.10 has a direct impact on the solution of the point density integral and therewith on the point render counters $N'_{i,j}$ that are estimated for each refinement pass. As mentioned at the beginning of section 4.1, R_i is updated on-the-fly in order to adapt to changes in the scene. Therefore, inaccurate time measurements may lead to deranging fluctuations of the $N'_{i,j}$ values. This effect occurs especially during constant interaction, when new input events force a restart of the refinement process just after the first pass. This problem is solved by storing the solution of the point density integral that has been utilized for the first refinement pass. It is reused for all following "first passes" as long as the visualization of the complete point object has not been completed once.

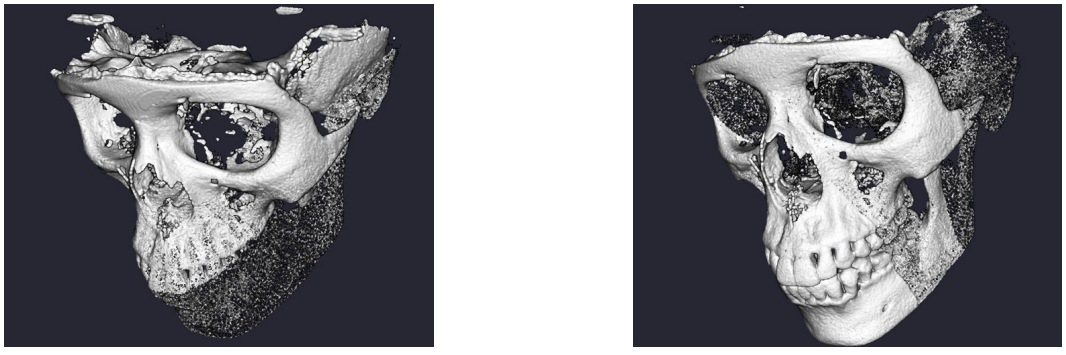


Figure 4.5: Large changes of the point density allocation during interaction.

Cascading refinement passes - Small values of the per-object target times t_i may result in a temporal "flickering" of the rendered model. The reasons are successively repeated updates of the visualization during slow interaction. This effect is noticeable especially if the display is refreshed after each rendering pass. Moreover, it is not covered by the solution for inaccurate time measurements. Therefore, a time delay during progressive refinement is incorporated. After visualization of the first pass, the initiation of subsequent refinement passes is delayed for ca. 150 ms.

Varying refinement priority - As described in section 4.1.1, the evaluation of the benefit functions for progressive refinement usually includes the utilization of view-dependent information. Therefore, the refinement priority may be subject to large variation, especially during fast interaction. This may effect in significant changes of the point render counters $N'_{i,j}$ for consecutive refinement passes. The solution to this problem is the application of a weighted blending of previous and current render count values for each bucket, i.e. $N'^{Blend}_{i,j} = N'^{Prev}_{i,j} \cdot w_{RC} + N'_{i,j} \cdot (1 - w_{RC})$. This blending has to be performed only for the first refinement pass. Notice that the weighted averaging of the render counters does not violate the cost constraint 4.3 due to $\sum_{k=1}^{M_i} N'^{Blend}_{i,k} = \sum_{k=1}^{M_i} N'_{i,k}$ for constant target render time t_i. Setting the user-defined render counter weight w_{RC} to zero will disable smooth update.

4.1.4 Experimental Results

In this section results for the point-based progressive refinement will be shown based on the point grid data structure. The same test environment and datasets have been used as in sections 3.1.4 and 3.2.3. Table 4.1 includes the per-frame initialization times of the adaptive rendering iterator for visibility-based refinement, as described in section 4.1.1. All measurements are based on a dry skull dataset at different volume resolutions and numbers of non-empty grid buckets (occupancy).

Dataset resolution	Grid resolution	Scalar range	Occupancy	Time
$512 \times 512 \times 512$	$4 \times 4 \times 4$	1900, 1900	48/64	1 ms
$512 \times 512 \times 512$	$4 \times 4 \times 4$	0, 4095	64/64	1 ms
$1024 \times 1024 \times 1024$	$8 \times 8 \times 8$	1900, 1900	253/512	10 ms
$1024 \times 1024 \times 1024$	$8 \times 8 \times 8$	0, 4095	512/512	17 ms
$1536 \times 1536 \times 1536$	$12 \times 12 \times 12$	1900, 1900	626/1728	92 ms
$1536 \times 1536 \times 1536$	$12 \times 12 \times 12$	0, 4095	1728/1728	174 ms

Table 4.1: Per-frame initialization times for visibility-based progressive refinement.

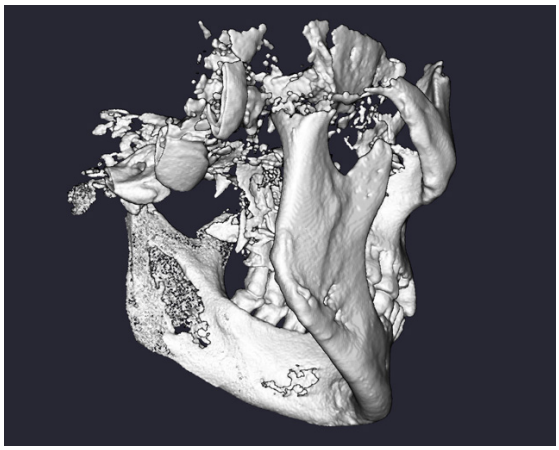
Timings include the determination of the refinement priority and calculation of point density. The initialization time is sufficiently low to achieve interactive visualization for lower point grid resolutions. It is just a small fraction of the total rendering time that includes splatting and shading. A significant slow-down has to be expected for higher point bucket counts. The reason is the naive implementation of the visibility-determination

algorithm based on cone-tracing with $O(n^2)$ time complexity, where n is the number of occupied grid buckets. A more optimized implementation, e.g. based on volume raycasting, should be considered in the future. Another possibility is the fallback to the depth-based refinement. Its initialization time is negligible, because it is implemented implicitly by a front-to-back traversal of the point grid structure.

In figure 4.6, the visual results for depth- and visibility-based refinement are presented. Each screenshot shows the first pass during surface- and MIP-based progressive rendering of a human dry skull for a volume resolution of 512^3 and grid resolution of 8^3. All images have been generated for an exponential point density function of second degree. The minimal point density has been set with respect to a minimal render count of 200K points. Visibility-refinement has a closer relation to "visual importance" for surface-based visualization, because structures with low density values are hidden behind point buckets with high densities. This effect is less useful for MIP-based visualization, because of the "look through" nature of volume-based rendering. Better results would be achieved by the incorporation of a rendering iterator, which is based on scalar intensities. Then, points with higher intensities could be associated with a higher priority for rendering [MGK99].

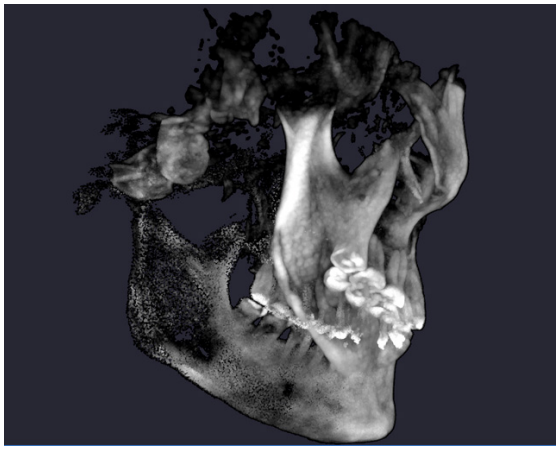
The diagrams in figure 4.7 illustrate the accumulated point densities during progressive refinement for four passes. The x-axis is associated with the per-frame rendering progress (the minimum value refers to the beginning of the first rendered bucket and the maximum to the end of the last one). The y-axis represents the corresponding point density (the maximum value refers to the full point count of a bucket). Each curve is associated with one refinement pass. Curves with a higher integral correspond to later passes (red-to-green color coding). All measurements have been performed for the *Head* dataset and a scalar range of $[1, 4095]$, consisting of ca. 19 millions of points (71.5 MB) and 24 occupied grid buckets. Rendering has been surface-based, including the utilization of a transfer function. A target frame rate of 30 FPS has been chosen for the first rendering pass. Subsequent refinement passes have been set up with 10 FPS. Constant curve segments illustrate the bucket structure of the point grid. Longer segments represent buckets with a high point count. Such irregularities are handled correctly by the presented progressive refinement framework using equation 4.8. The exponential function exhibits a lower maximum point density for the first rendering pass. Its average gradient magnitude is lower, effecting in smoother transitions among bucket borders. Exponential functions with relative low degree values should be chosen, if a regular point density distribution with a low per-bucket variation is desired. The polynomial function exhibits a higher maximum point density for the first pass at the cost of a steeper progression. It can be used - together with a high degree value - if the user has a predominant attention on the high priority buckets.

Table 4.2 shows the progress of the frame rendering rates during refinement of extended surface models with transfer function look-ups. Included are the numbers of refinement passes as well as the minimal and maximal rates. Partial rendering passes have not been considered. The target frame rate has been set to 25 FPS. A more reasonable approach is to choose a significantly lower value (e.g. 3 FPS) for subsequent refinement passes. Such a strategy assures faster convergence of the visualization and a significantly lower count of refinement passes. The maximal relative error for the minimal frame rates is

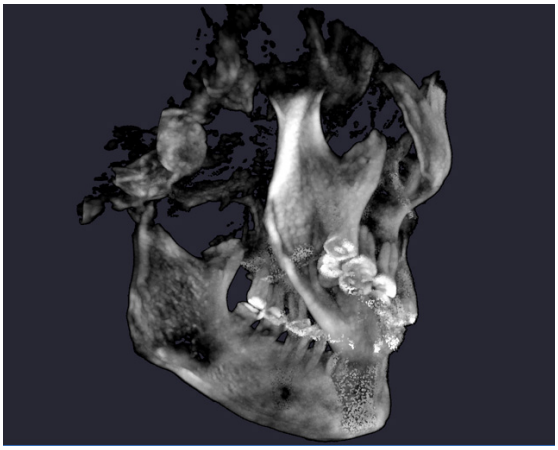


(a) Depth-based refinement (Surface)

(b) Visibility-based refinement (Surface)



(c) Depth-based refinement (MIP)



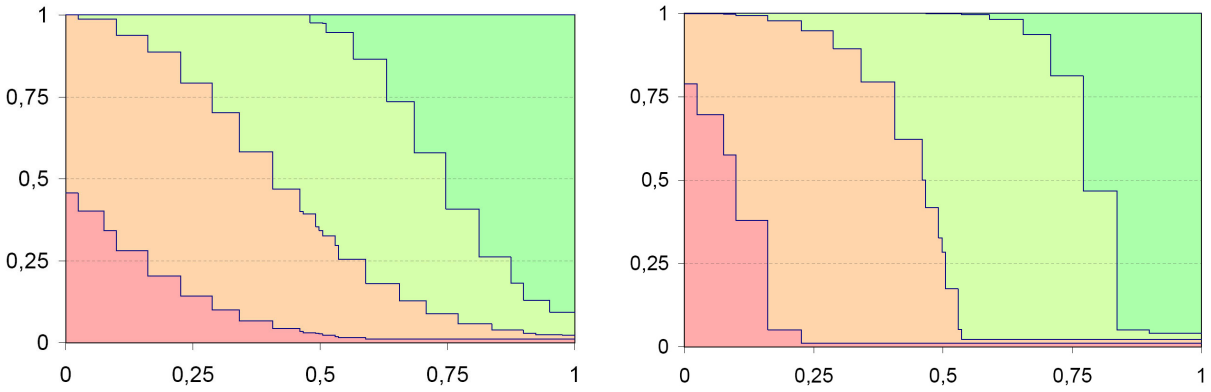
(d) Visibility-based refinement (MIP)

Figure 4.6: Comparison of different strategies for point-based progressive refinement.

ca. 5 % and approximately 8 % for the maximal values. The reason for these variations are inaccuracies during on-the-fly estimations of the rendering rates and solutions of the point density integral. Nevertheless, the results are reasonable for practical utilization. Small variations of the frame rate during interaction are usually not noticeable for the user. Comparable timings have been measured for MIP and visibility-based refinement.

Dataset	Scalar range	Points	Passes	Minimal	Maximal
Abdomen	1125, 4095	5,937,926	3	24.6 FPS	25.8 FPS
Dry Skull	1900, 4095	8,974,213	4	24.1 FPS	25.4 FPS
Legs	2714, 4095	28,061,756	12	23.8 FPS	26.7 FPS
Visible Female	781, 4095	81,856,959	35	24.3 FPS	27.0 FPS

Table 4.2: Rendering rates during progressive refinement with depth-based priority.



(a) Exponential point density (Degree 2)

(b) Polynomial point density (Degree 8)

Figure 4.7: Accumulated point density curves in relation to per-frame rendering progress. The examples are shown for four refinement passes. The total integral values of each curve (colored red to green) are proportional to the number of points rendered in each pass.

4.2 Sequential Decomposition of Point Objects

An important goal has been the development of a pure sequential LOD technique, i.e. the detail level should only be controlled via the prefix length of a point array. The first reason is a simple point density control, which can be integrated into the aforementioned progressive refinement framework. Moreover, the natural layout of computer memory is sequential, so sequentialization results in cache-efficient processing of large datasets using *block operations* [PSL05]. Additionally, linear data structures facilitate rapid development of simple data generation and rendering components.

The main challenge is the same as for "static" point-based visualization, i.e. a watertight representation without holes is essential for a consistent impression of shape. The solution in this work follows Hill's et al. approach for progressive image transmission, who used "fat pixels" for the 2D visualization of coarse images [HWG83]. Similarly, the 3D analog is incorporated ("fat points"), by increasing the splat size for lower point densities.

One well-known example for sequential multiresolution rendering of point models is Dachsbacher's et al. *sequential point trees* technique, including derivatives like Pajarola's et al. *XSplat* and Wimmer's et al. *Instant Points* [DVS03, PSL05, WS06]. Most of these approaches store also "interior nodes" of an associated hierarchy, effecting in an additional processing and memory overhead as well as per-point LOD control operations. An exception is Wimmer's et al. recent rendering system for laser range scanning datasets. Their *memory efficient sequential point trees* data structure shows some important conceptual aspects of a pure sequential LOD technique. They avoid storage of interior hierarchy nodes by reusing existing child nodes as representatives for the ancestors. Similarly to the approach in this work, the GPU needs to process only a prefix of the point list, not requiring any additional per-primitive operations during rendering. Wimmer et al. require an extra index array for the management of hierarchical levels. It has to be processed by the CPU

for every rendering frame. All mentioned approaches have a "hierarchical origin", effecting in a coarse LOD granularity and relatively complicated processing.

In this work, the LOD generation step is interpreted as a reordering of the original point set. The final multiresolution representation consists of exactly the same primitives as the input model. There is no need for the calculation of extra "average information". Moreover, there are no explicit detail levels at all, effecting in a fine multiresolution granularity. Let $P = \{p_1, \ldots, p_N\}$ with $p_i \in \Re^3$ be a set of points, e.g. within a bucket of a high-level data structure. Assume that each element is considered as a geometric primitive in 3D with constant radius $r \in \Re$. A sequential decomposition of P consists of two sequences P' and R with the following properties:

- $P' = (p_{i_1}, \ldots, p_{i_N})$ is a permuted point sequence of P
- $R = (r_1, \ldots, r_N)$ is a radius sequence with $r_i \geq r$ for $1 \leq i \leq N$
- $\forall N' \in \{1, \ldots, N\}, \forall p_i \in P, \exists p_{i_j} \in P'_{N'} : D(p_i, p_{i_j}) \leq r_{N'} - r$

$P'_{N'}$ represents the prefix sequence of P' with length N'. $D : \Re^3 \times \Re^3 \rightarrow \Re$ is the Euclidean distance function in 3D. The last key point is called *covering criterion*. It can be interpreted as follows: Consider any prefix of P' of length $N' \leq N$. Assume that each element in this prefix is associated with the N'-th radius in R. Then, the N' points in the prefix have to "cover" all N points in P.

In the remainder of this section different approaches for the computation of the point and radius sequences will be provided. Finally, their integration into the presented universal rendering system will be discussed.

4.2.1 Reordering Algorithms

Using an arbitrary permutation of the initial point set for the generation of the sequential decomposition could effect in an unbalanced point distribution. Then, excessively large point sizes would be required to satisfy the aforementioned covering condition, resulting in a low-quality LOD representation and an unnecessary rasterization overhead during rendering. Therefore, the reordering procedure has to consider the spatial distribution of the point primitives.

The main idea for the computation of the sequential decomposition is the maximization of the mutual point distances for all prefixes of the point sequence. Advantage is taken of the *Hausdorff distance* for this purpose. This metric has been utilized successfully for a variety of computer science problems, like pattern recognition, registration and as geometric error metric for mesh simplification [KLS96]. Using it for the calculation of the radius sequence will assure compliance with the covering criterion and minimal point sizes. On the one hand, such an approach considers only positional information during LOD generation. It cannot take advantage of more sophisticated detail criteria, like surface curvature. On the other hand, its simplicity and generality is its strength. It can be applied to surface and volume models, supporting the vision of a universal visualization pipeline in this work.

Straight-Forward Algorithm

Let $D_{min} : \Re^3 \times Q \rightarrow \Re$ be the minimal distance between a point p and a point set Q in 3D space, i.e.: $D_{min}(p, Q) = min_{q \in Q}\{D(p, q)\}$. D_{max} is defined analogously to D_{min}. Moreover, let $H : P \times Q \rightarrow \Re$ be the asymmetric Hausdorff distance from point set P to Q, i.e. $H(P, Q) = max_{p \in P}\{D_{min}(p, Q)\}$. Then, a *Hausdorff point subset* $H_Q^P \subseteq P$ with respect to set Q is defined as follows:

$$H_Q^P = \{\ p \in P \mid D_{min}(p, Q) = H(P, Q)\ \} \tag{4.14}$$

The value $H(P, Q)$ that is associated with each $p \in H_Q^P$ is called the *Hausdorff point distance* for H_Q^P. Algorithm 5 illustrates a straight-forward approach for the calculation of a sequential decomposition (P', R) for the point set $P \neq 0$. It is assumed that the sequences P' and R are empty at the beginning.

Algorithm 5: Straight-forward computation of a sequential decomposition.

```
1 Remove an arbitrary point p ∈ P and append it to P';
2 Append D_max(p, P) + r to R;
3 while P ≠ 0 do
4 |   Select an arbitrary point p ∈ H_P'^P;
5 |   Remove p ∈ P and append it to P';
6 |   Append H(P, P') + r to R;
7 end
```

The main idea of this algorithm is the iterative picking of a point, which is currently associated with the Hausdorff point distance $H(P, P')$. The Hausdorff point subsets $H_{P'}^P$ represent the candidate point sets for P' at each iteration step. Using $H(P, P')$ as the point size offset for the points in P' results in a covering of all original elements in P. In other words, the Hausdorff point distances are used to derive the radii for the "fat points" in the sequence R.

For the selection of an arbitrary point from P it is suggested to select the point with the smallest distance to the bounding box center of the source point set. Such a strategy minimizes the current Hausdorff point distance and therewith the point radius in sequence R. A sample result of algorithm 5 is illustrated in figure 4.8. The straight-forward computation of the sequential decomposition has a time complexity of $O(N^3)$. It is far too slow for practical utilization, especially for datasets with millions of point primitives.

The iterative selection of a point from a Hausdorff point subset reminds of Eldar's et al. *Farthest Point Sampling* (FPS) algorithm for progressive image sampling [ELPZ97]. Their approach is based on the following idea: "Place the next image sample at the center of the least known region". The authors have shown that such a strategy assures a uniform sample distribution and has excellent antialiasing properties. As mentioned in section 2.2, Moenning et al. have utilized the FPS strategy for the simplification of point-based surfaces [Moe06]. In contrast to this work, they have assumed that the input point data

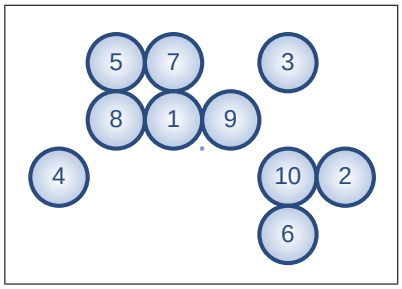


Figure 4.8: Example of a sequential decomposition. The numbers represent the element order. The light blue dot illustrates the bounding box center of the point set.

is uniformly distributed and embedded into a Cartesian grid. Their algorithm is based on the construction of a bounded Voronoi diagram and has a worst-case complexity of $O(N^2 \cdot log(N))$ [MD04]. Moenning et al. have not discussed the utilization of their irregular sampling approach for sequential multiresolution rendering in the context of progressive refinement.

The investigations of this work concentrate primarily on the classical hole-filling problem of point-based LOD visualization. Simple and efficient solutions for the computation of the sequential decomposition will be provided in the remainder of this section. All algorithms can be executed directly on the point dataset, not requiring the explicit generation of an auxiliary data structure.

Deflating Spheres

The first optimization for the computation of the sequential decomposition is based on the following geometric interpretation of the problem: Consider P' as a sequence of spheres, associated with the radii in R. Longer prefixes of the sequence will imply higher point densities and result in smaller sphere sizes that are required to fulfill the covering criterion. Therefore, all spheres in P are initially associated with a large radius and it is decreased by $\Delta r \in \Re$ as long as all points are covered. Then, all uncovered points are inserted into the output sequence P' and the process is repeated until a minimum radius $r_{min} \in \Re$ is reached. This idea is realized in algorithm 6.

The first three iterations of the algorithm are illustrated in figure 4.9. Let p_0 be the first point that is removed from P and inserted into P'. Moreover, let $M = \lceil \frac{D_{max}(p_0,P)+r-r_{min}}{\Delta r} \rceil$. Then, algorithm 6 has a time complexity of $O(M \cdot N^2)$. Typically, M is much smaller than N. Nevertheless, a quadratic complexity with respect to the point count may not be acceptable for high-resolution datasets.

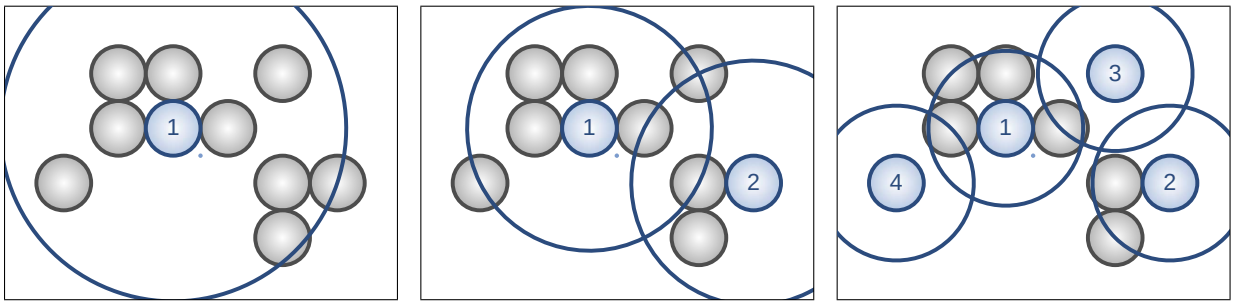
Binary Space Partitioning

A further optimization of the sequential decomposition procedure can be achieved by reducing the number of coverage tests. The main idea is to avoid unnecessary computations

Algorithm 6: Computation of a sequential decomposition via deflating spheres.

```
1  Remove an arbitrary point p ∈ P and append it to P';
2  r_H ← D_max(p, P) + r;
3  Append r_H to R;
4  while r_H ≥ r_min do
5  |  foreach p ∈ P do
6  |  |  if D_max(p, P') + r < r_H then
7  |  |  |  Remove p ∈ P and append it to P';
8  |  |  |  Append r_H to R;
9  |  |  end
10 |  end
11 |  r_H ← r_H − Δr;
12 end
13 Append all remaining points in P to P';
14 Associate all these points with r_min in R;
```

for distant points in the inner loop of algorithm 6. For example, consider its second iteration, i.e. figure 4.9(b). There, point $p_4 \in P$ is close to $p_1 \in P'$ and therefore requires no coverage check with respect to $p_2 \in P'$. Therefore, one can separate the point set P into two subsets, one containing all points that are closer to p_1 and the other one all points closer to p_2. Then, the maximum distance tests in line 6 can be performed recursively for each subset. Such a separation can be realized by applying a *binary space partitioning* (BSP) on the input set P. Similarly to the trivial construction of a 3D Voronoi cell for two points, the partitioning plane is placed between p_1 and p_2. The corresponding plane equation $E^{p_1}_{p_2} : \Re^3 \rightarrow \Re$ for point p_1 and p_2 can be written as $E^{p_1}_{p_2}(p) = D(p_1, p)^2 - D(p_2, p)^2$. The BSP-based approach for the computation of the sequential decomposition is presented in algorithm 7.



(a) First step (b) Second step (c) Third step

Figure 4.9: First three iterations of the deflating spheres algorithm. The blue points represent the active elements, which are used for the covering of the complete point set in the corresponding step.

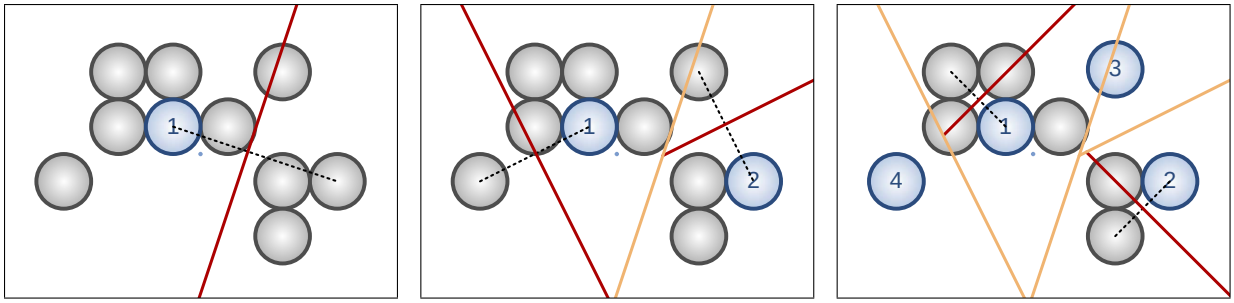
Algorithm 7: BSP-based computation of a sequential decomposition.

begin *Main*
  Remove an arbitrary point $p \in P$ and append it to P';
  $r_H \leftarrow D_{max}(p, P) + r$;
  Append r_H to R;
  BinarySpacePartition(p, P);
  Sort all elements in P' and R in decreasing order of radius;
end
begin *BinarySpacePartition* : point $p'_1 \in P'$, subset $Q \subseteq P$
  $d_{max} \leftarrow D_{max}(p'_1, Q)$;
  Select an arbitrary point $p'_2 \in \{\ q \in Q \mid D(p'_1, q) = d_{max}\ \}$;
  Remove $p'_2 \in Q$ and append it to P';
  Append $d_{max} + r$ to R;
  $Q_1 \leftarrow \{\ q \in Q \mid E^{p'_1}_{p'_2}(q) < 0\ \}$;
  $Q_2 \leftarrow Q - Q_1$;
  BinarySpacePartition(p'_1, Q_1);
  BinarySpacePartition(p'_2, Q_2);
end

The elements of the resulting sequences P' and R are not in the correct order due to the recursive partitioning of P. Therefore, a final sorting step is required using the values of R as the sorting keys. This can be realized efficiently using e.g. the radix sort algorithm with linear time complexity [Sed98]. The first three iterations of the algorithm are shown in figure 4.10.

On the one hand, the algorithm has a time complexity of $O(N \cdot log(N))$ and is therefore efficient enough for practical utilization, even for complex medical models. On the other hand, due to its greedy nature it generates results only at lower quality compared to algorithms 5 and 6. The reason is the independent processing of the partitioned point subsets during recursion. As illustrated in figure 4.11, this may lead to a failure of the maximum distance check for points that have been allocated in two different subsets. It may result in unnecessary large point sizes that are apparent especially at lower detail levels.

Although, it is not required to create an explicit hierarchical representation of the input point set, the binary splitting strategy of algorithm 7 is similar to Rusinkiewicz's et al. *recursive splitting* and Pauly's et al. *hierarchical clustering* [RL00, PGK02]. Nevertheless, the authors define the binary split along the longest axis and direction of greatest variation, respectively. In contrast to this work, they have not focused on a sequential LOD representation, including an on-the-fly computation of point sizes for covering rendering primitives that correspond to finer resolution levels.



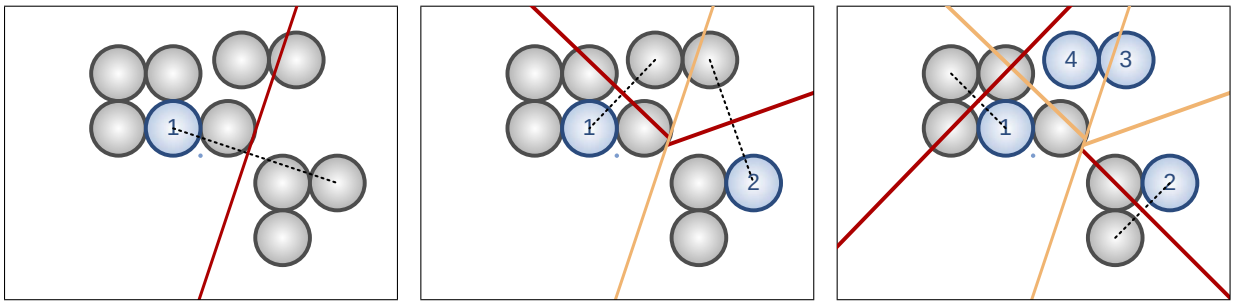
(a) First step (b) Second step (c) Third step

Figure 4.10: First three iterations of the BSP-based algorithm. Current partitioning planes are colored in red. Planes from previous iterations are colored in orange. The black dotted line illustrates the maximum distance for the corresponding BSP-cell.

4.2.2 Visualization with Adaptive Point Sizes

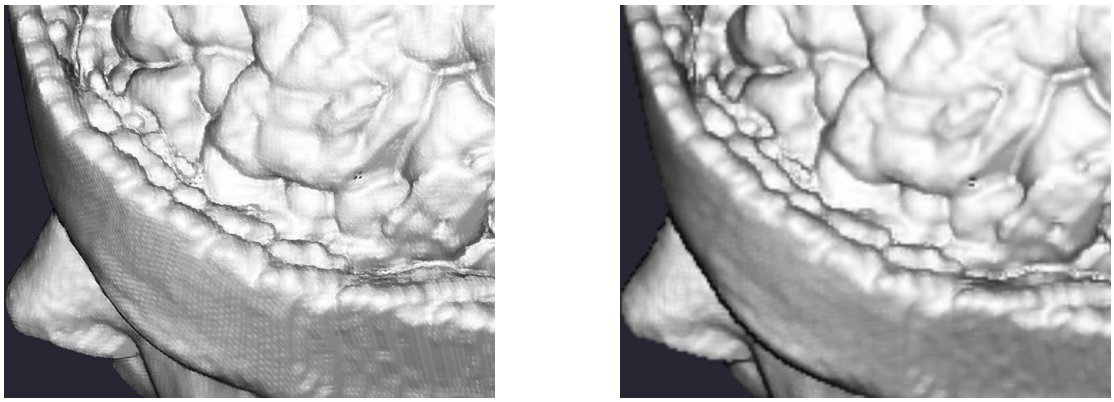
The presented progressive refinement framework is based on a bucket-wise point density allocation. Using the original point radius r for array-based rendering of lower details levels would produce holes during the visualization. Therefore, points are rendered with an increased radius $r_{N'}$ (the N'-th radius from sequence R) for buckets with render point count $N' < N$. Such an approach assures a watertight rendering due to the covering criterion of the sequential decomposition.

Usually, many values in R are identical, especially in the context of structured point datasets. In order to reduce the memory overhead of the radius sequence, *run-length-encoding* is applied by storing a sequence of tuples $(r_1, i_1), \ldots, (r_K, i_K)$ with $r_k \in R$ and $i_k = min\{\ 1 \leq i \leq N \mid r_i = r_k\ \}$. The index i_k refers to the first radius in R with value r_k. The tuples can be used to look-up $r_{N'}$ with O(log(K)) time complexity for each bucket via binary search.



(a) First step (b) Second step (c) Third step

Figure 4.11: Failure of the maximum distance check for the BSP-based algorithm. In the second iteration, it is not executed for points 3 and 4, effecting in the generation of a value greater than the Hausdorff point distance.



(a) Original (b) Viewport scaling enabled

Figure 4.12: Smoother visual appearance as a positive side-effect of viewport scaling.

A crucial aspect of splatting with adaptive point sizes is the associated rasterization overhead per point primitive. This can have a "feedback effect" on the adaptation of the point rendering rate in equation 4.4 and therewith on progressive refinement control. In the worst case, e.g. during high magnification, the performance gain resulting from a decreased point count may be superimposed by the excessive drawing of pixels. The solution to this problem is based on the tradeoff between image quality and interactivity. At the beginning of each refinement pass, the original viewport resolution is downscaled uniformly based on current viewing parameters. A similar idea has been realized by Kawata et al. for z-buffer gradient shading [KK04]. Their motivation is the reduction of the maximal splat diameter to the size of one pixel in order to facilitate the normal vector calculations in image space. Such an aggressive reduction degrades significantly visual quality, especially in regions with fine detail.

For the purposes of this work it is enough to estimate the downscaling factor based on a weighted average between the minimal and maximal expected splat size, i.e. $d_{min} \cdot w_V + d_{max} \cdot (1 - w_V)$. During interaction, smaller values of weight $w_V \in [0, 1]$ are chosen to assure interactivity. Additionally, the downscaling factor is blended with the previously estimated value in order to reduce "flickering" artifacts. The expected splat sizes $d_{min}, d_{max} \in \Re$ are approximated by picking a near and a far point on the bounding volume of each bucket. Projecting these points based on to the adaptive radius $r_{N'} \in R$ from the previous refinement pass results in an estimation of the corresponding pixel sizes. The visual effect of viewport resolution scaling is illustrated in figure 4.12. Notice that it may improve interactivity also for "static" visualization, e.g. for low-resolution models and large splats during magnification.

4.2.3 Experimental Results

In this section results for the level-of-detail rendering based on sequential decomposition of point datasets will be presented. Refer to sections 3.1.4 and 3.2.3 for the information about

the test environment and datasets. Table 4.3 includes the timings for the computation of the sequential decomposition using the deflating spheres (DS) algorithm and the version based on binary space partitioning (BSP). Occupancy refers to the number of non-empty point grid buckets.

Dataset	Scalar range	Occupancy	Points	DS time	BSP time
Head	1077, 1077	24/24	541,620	00:00:30	0.3 s
Vessels	575, 4095	43/64	1,551,100	00:03:21	0.9 s
Abdomen	1125, 4095	75/96	5,937,926	00:26:40	3.2 s
Dry Skull	1900, 4095	92/216	8,974,213	00:37:10	4.8 s
Legs	2714, 4095	75/100	28,061,756	07:40:55	16.2 s

Table 4.3: Computation times for sequential decomposition.

The point reordering routines are executed bucket-wise. The aforementioned operation data structure is based on structured points, making a completely integer-based implementation possible. Grid buckets with a high point count have a significant impact on the overall computation time, especially for the deflating spheres algorithm with its quadratic time complexity. It is 2-3 orders of magnitude slower than the BSP-based version and practically not applicable in a typical medical environment. Further optimizations are required. The BSP algorithm performance is adequate for medical applications. The processing time is significantly lower than half a minute, even for high point counts. In the remainder of this section, the focus is on the quality comparison between both approaches.

Table 4.4 shows the percentages of rendered points for the first pass during interaction and a target frame rate of 25 FPS. Viewport scaling, as described in section 4.2.2, has been temporarily disabled. Included are results for "thin" and "fat points". The former relates to primitives that are rendered at the original point size $r \in \Re$. The latter refers to points with extended sizes $r_i \in R$, required to close holes in lower detail models. Measurements have been performed separately for surface and volume rendering.

Dataset	Scalar range	Points	Type	Thin	Fat DS	Fat BSP
Abdomen	1125, 4095	5,937,926	Surface	27.4 %	14.0 %	4.6 %
Dry Skull	1900, 4095	8,974,21	Surface	22.5 %	10.7 %	2.5 %
Legs	2714, 4095	28,061,75	Surface	5.9 %	3.7 %	0.1 %
Abdomen	1125, 4095	5,937,926	MIP	15.7 %	1.8 %	< 0.1 %
Dry Skull	1900, 4095	8,974,21	MIP	13.3 %	1.0 %	< 0.1 %
Legs	2714, 4095	28,061,75	MIP	3.7 %	< 0.1 %	< 0.1 %

Table 4.4: Rendering of fat points for the first rendering pass during interaction.

Higher percentage values are favorable, because the user gets a more detailed image after the first rendering pass. Rendering with fat points effects in significantly lower percentage values compared to the visualization with original sizes (thin points). A similar relation can be observed between the BSP-based LOD generation and the deflating spheres algorithm.

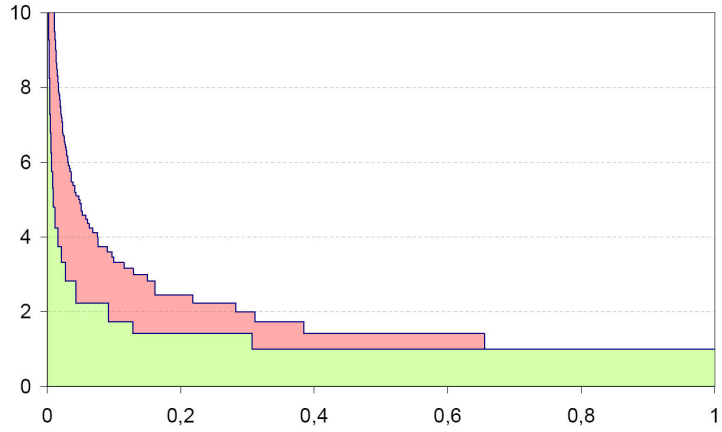


Figure 4.13: Point size offset curves computed during sequential decomposition. The lower curve refers to the deflating spheres algorithm, the upper one to the BSP-based version.

The reason is the rasterization overhead for large splat sizes. This problem is solved by the on-the-fly adaptation of the viewport resolution. Corresponding results for a downscaling weight of $w_V = 0.5$ are presented in table 4.5.

Dataset	Scalar range	Points	Type	Thin	Fat DS	Fat BSP
Abdomen	1125, 4095	5,937,926	Surface	35.7 %	40.0 %	37.7 %
Dry Skull	1900, 4095	8,974,21	Surface	25.4 %	26.6 %	23.1 %
Legs	2714, 4095	28,061,75	Surface	7.7 %	8.5 %	6.7 %
Abdomen	1125, 4095	5,937,926	MIP	23.9 %	37.2 %	20.8 %
Dry Skull	1900, 4095	8,974,21	MIP	18.6 %	25.4 %	13.3 %
Legs	2714, 4095	28,061,75	MIP	5.7 %	5.8 %	1.0 %

Table 4.5: Rendering of fat points with an adaptive viewport resolution.

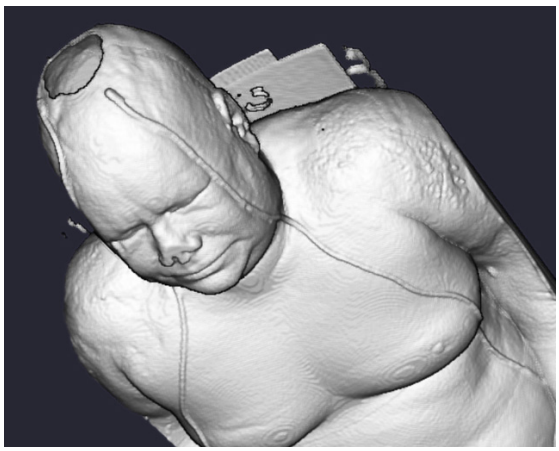
The adaptation of the viewport resolution compensates satisfactorily the performance degradation for fat splat rendering. Throughout higher percentage values have been measured, even for rendering with the original splat sizes. The best results have been achieved for point models that have been generated using the deflating spheres algorithm. All evaluations have been performed with circular splat shapes. Up to 25 % higher point counts have been observed for rectangular primitives, so a promising extension would be the temporary adjustment to a simpler splat shape for the first rendering pass during interaction.

Figure 4.13 shows a point size diagram for a grid bucket of a dry skull dataset with 132,813 points. The lower curve corresponds to the deflating spheres algorithm, the other one to the BSP-based approach. The x-axis is associated with the fraction of points for rendering. The y-axis represents the point size offsets that are required to close the holes in the 3D model (for example, the value 2.0 is associated with twice the original size).

The point size offset curves can be utilized for the derivation of the sequential decomposition quality. Lower values represent a better LOD quality, associated with smaller splat

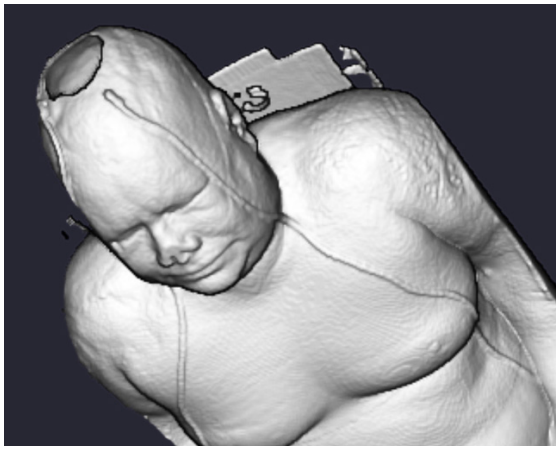
sizes for rendering. The minimal point offset is achieved at a point fraction of ca. 30.7 % for the deflating spheres algorithm and 65.5 % for the BSP-based version. This is more than "twice as good" at the cost of significantly higher preprocessing times. A simple quantitative estimation of the sequential LOD quality for a single point bucket is the integral of the corresponding point size offset curve. Assuming that the deflating spheres algorithm provides the optimal solution for the sequential decomposition, it is possible to estimate the relative approximation error for the BSP-based approach (represented as the red colored area in the diagram) [ELPZ97]. The deflating spheres approach achieves an integral value of ca. 1.32 and the BSP-based version of 2.02. The resulting relative error is approximately 53 %. The maximal point size offset has been ca. 108.2 for both cases. Its bound is the radius of the bounding sphere of the corresponding point bucket. It is $\sqrt{3 \cdot ((2^7 - 1)/2)^2} \approx 110.0$ for a point coordinate quantization based on seven bits. One can observe that the shapes of the point size offset curves with a high point count are very similar for different datasets and scalar ranges. Therefore, a promising issue for further work is the development of an analytical curve description.

Figure 4.14 shows the visual results for the first rendering pass, including point-based LOD rendering with enabled viewport scaling (for a weight of $w_V = 0.5$) compared to standard raycasting. The point density function has been exponential of degree two for the surface-based visualizations, and it has been constant for the MIPs. The isovalue of the *Visible Female* dataset is 781. The scalar range for the *Abdomen* dataset is [1125, 4095]. On the one hand, traditional multiresolution raycasting requires no additional preprocessing, i.e. adaptive rendering can be activated on-the-fly. On the other hand, the irregularity and adaptivity of the point-based visualization result in significantly better visual quality. The raycaster could be further improved by using advanced 2D interpolation techniques. Nevertheless, it requires the reduction of the target frame rates (3-6 times lower than for point-based rendering) in order to produce images of tolerable quality. The visual difference between the deflating spheres algorithm and the BSP-based approach is hardly noticeable. The point models using BSP-based LOD generation appear slightly more blurred. Such a result does not justify the excessively higher preprocessing times of the deflating spheres algorithm.



(a) Deflating spheres (30 FPS)

(b) Deflating spheres (30 FPS)



(c) BSP-based (30 FPS)

(d) BSP-based (30 FPS)

(e) Raycasting (5 FPS)

(f) Raycasting (10 FPS)

Figure 4.14: Coarse LOD rendering of *Female* and *Abdomen* datasets.

Chapter 5

Implementation and Examples

The focus in this chapter is on implementation-related details of the aforementioned point-based techniques for medical visualization. One aspect is the brief overview of the underlying software architecture. Another aspect is the presentation of the resulting application examples, including a simple viewer application for medical datasets, a progressive cutting tool for complex point models and a clinical planning system for maxillo-facial distraction osteogenesis.

All developed techniques that have been presented in chapters 3 and 4, including point-based exploration, universal rendering, adaptive refinement and sequential decomposition, have been incorporated into a software library, called *PointPack*. The corresponding software architecture is based on the Julius software platform [JDH+05]. It is a component-based framework for rapid development of visualization-related applications, including scientific prototyping and commercial product development. Its main characteristics are modularity, extendibility and reusability.

The high-level design of PointPack is based on the so-called *component infrastructures*. They can be considered as "meta software patterns", facilitating application design and providing base components with fundamental functionality and predefined core interfaces. The following infrastructures have been most relevant for the incorporation of the aforementioned point-based techniques:

Data-processing - Provides data components for encapsulation of information. They are created and manipulated by algorithmic components.

Visualization - Includes components for rendering and scene management. A scene is represented by a visualization component, associated with one or more renderers.

User-interface - Comprises a configurable graphical user interface (GUI) for the implementation of workflows, views and file management.

Most functionality of the PointPack library has been realized in the data-processing and visualization layers. The corresponding main component architectures are illustrated in figures 5.1 and 5.2 using high-level UML class diagrams.

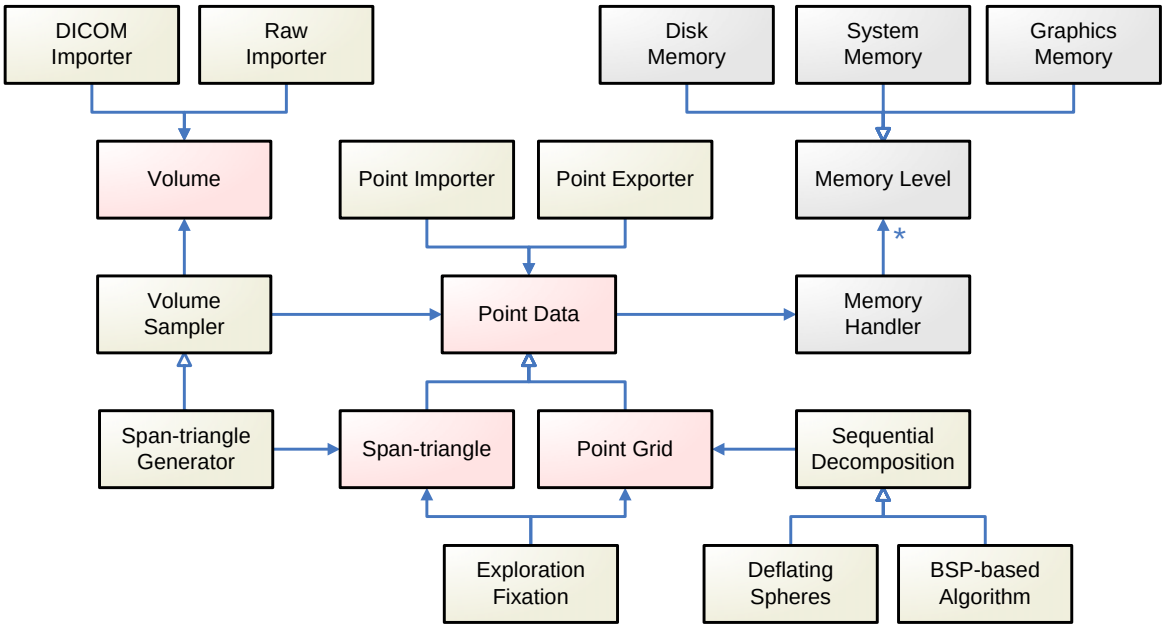


Figure 5.1: Data-processing architecture. Data components are colored in red, algorithms in yellow and general-purpose functionality in gray.

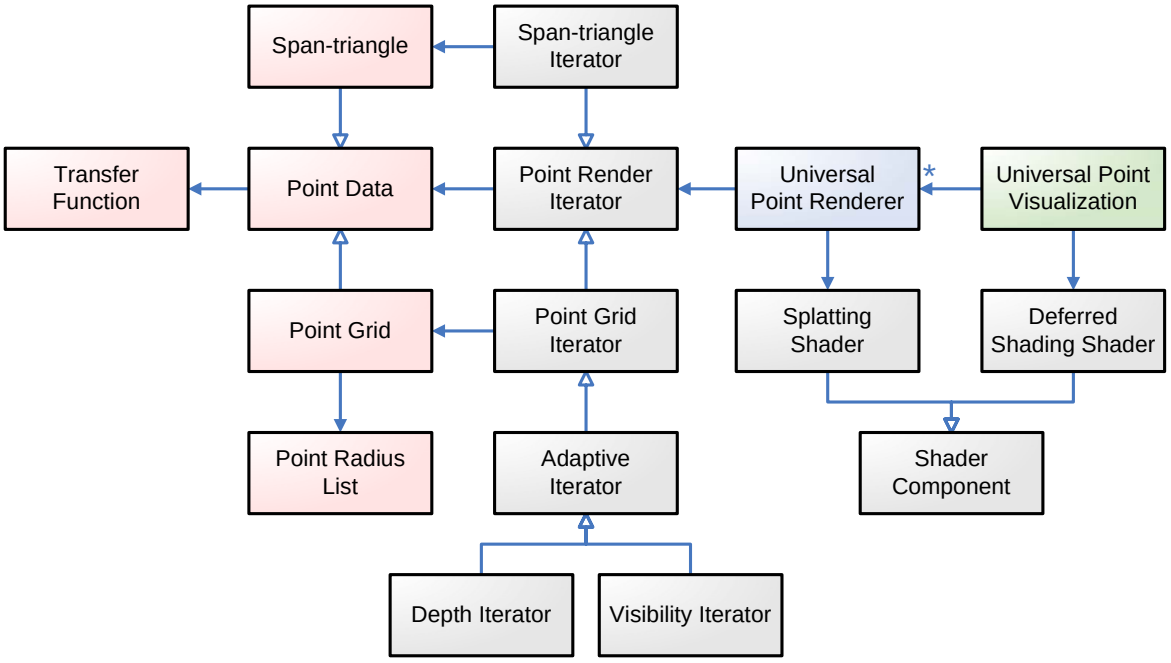


Figure 5.2: Visualization architecture. The same coloring is used as in the previous figure. Additionally, rendering and visualization components are in blue and green, respectively.

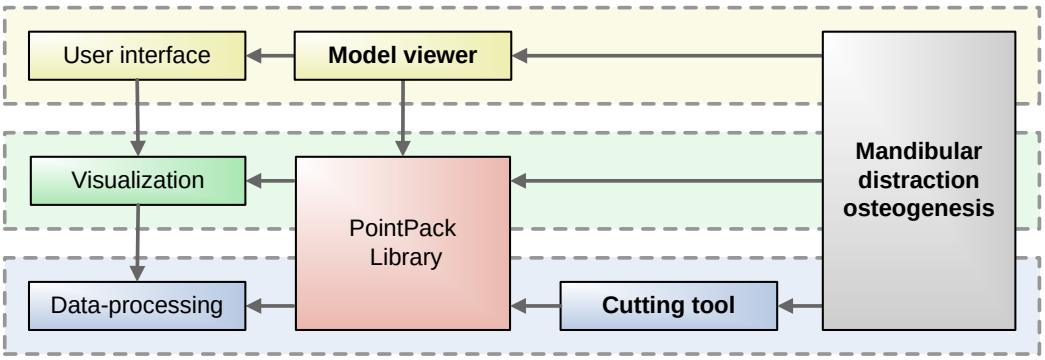


Figure 5.3: Relationships between point-based application examples. The point-based model viewer, progressive cutting tool and the mandibular distraction osteogenesis application will be presented in the remainder of this chapter.

The visualization architecture takes advantage primarily of the data-processing components, like the point data structures and associated iterators. The class diagrams summarize the main aspects of the work from a software-engineering point-of-view, including:

- Memory system for stable out-of-core allocation of large datasets [vRLK05]
- Basic point data structures, including abstract data iterators
- Algorithmic components for the generation and conversion of point data
- Universal point-based rendering component
- OpenGL-based GPU shader tools for deferred splat shading
- Progressive rendering iterators for adaptive traversal of point data.

PointPack has been used for the implementation of the examples that will be presented in the remainder of this chapter. As illustrated in figure 5.3, a common component pool is used for the development of applications that are realized in different software projects. It comprises all three aforementioned infrastructures. The PointPack library will be made public available on the website `http://www.julius.caesar.de`

5.1 Point-based Viewer Application

The first example application is a viewing tool for three-dimensional models based on a purely point-based visualization pipeline. It can be considered as a basic demonstration of the techniques that have been developed in the context of this work. On the one hand, end-users can use the viewer for the import and interactive exploration of volumetric datasets, including the option on saving and restoring of the current visualization session. On the other hand, developers can take advantage of PointPack's component architecture and

extend the viewer application with own functionality, e.g. by incorporating additional file formats and navigation devices.

The integral part of the viewer's GUI is the *application center*, consisting of several Julius user-interface components. It can be used for the control of the point-based processing pipeline as well as for the adjustment of rendering and progressive refinement parameters:

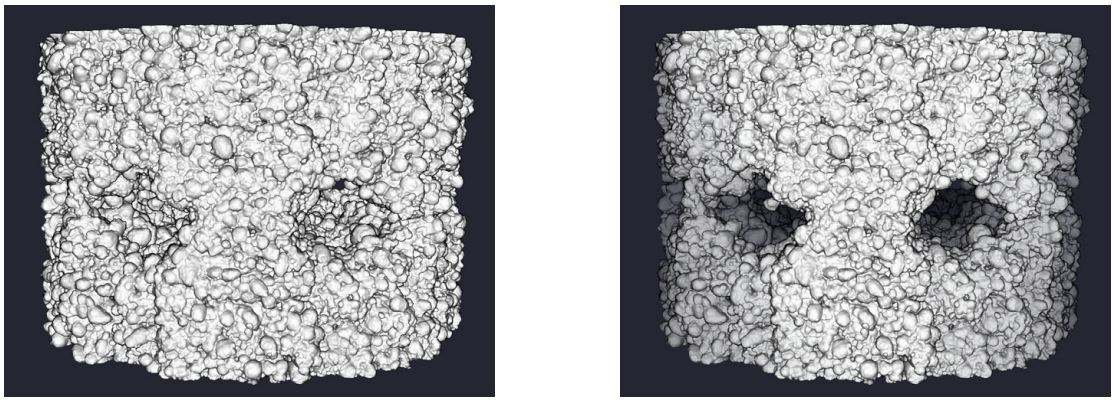
- **Generation -** Setting of exploration range, volume sampling quality and building up of exploration data structure (Sections 3.1.1 and 3.1.2).
- **Exploration -** Adjustment of exploration parameters, fixation of point model and conversion to operation data structure (Sections 3.1.1 and 3.1.2).
- **Cutting plane -** Activation of 3D cutting plane, setting of its position, orientation and "thickness" (Section 3.2.1).
- **Transfer function -** Creation of default transfer functions, transfer function editor and parameters for maximum-intensity-projection visualization (Section 3.2.1).
- **Rendering -** Selection of splat shape, quality of normal estimation technique, color filtering and shading parameters as well as depth attenuation (Section 3.2.2).
- **Adaptivity -** Setting of target frame rate, type of point density function and adaptation of viewport resolution (Sections 4.1.1, 4.1.2 and 4.2.2).

The viewer application can be also used for the visualization of non-medical datasets. Figure 5.4 shows the screenshot of a micro-CT scanned rapid prototyping (RP) model, representing a ceramic scaffold that has been created by a 3D printing technique. The utilization of edge enhancement, depth attenuation and cutting planes has proven to be especially useful for the understanding of such complex structures [CGPD98]. Other non-medical volume datasets are shown in figure 5.5.

Viewer Application Design

Several design advantages that result from the universal rendering approach have been considered during the development of the viewer application. In contrast to the traditional techniques, it makes it possible to generate slice, surface and volume visualizations with one highly configurable renderer component. Such an approach results in a simpler application design and control, especially for the following functionality: propagation of visualization settings, handling of the renderer activation states and synchronization of updates among different visualizations.

Another aspect is the facilitated user interface design. Traditional medical GUIs consist of many different elements, e.g. one 3D view and several 2D slice views (figure 1.2). The separated display is usually considered as effective for expert users, but it may be hard to control for non-experts. Inspired by one of Preim's golden rules for the development of



(a) Original (b) Depth attenuation enabled

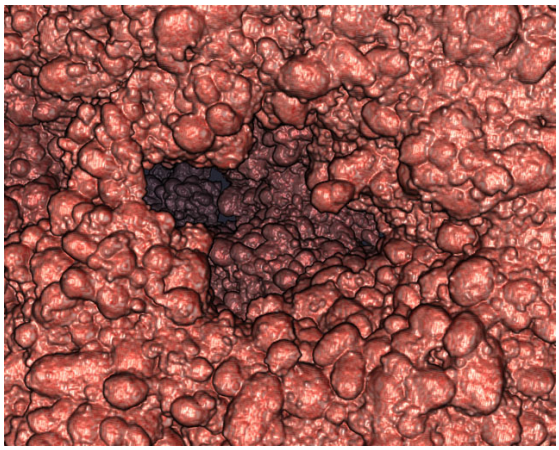
Figure 5.4: Point-based isosurface of a large micro-CT RP model (ca. 52.6 mio. points).

smart medical applications, universal rendering makes it possible to combine directly various visualization types in one view of the GUI [PP03]. Such an approach makes it possible to concentrate the user focus and automatically preserve context information (e.g. spatial correspondence), even if switching between different display configurations. Additionally, it facilitates the development of the user interaction system for the virtual models in the scene, because only one view with a single renderer class has to be considered during design.

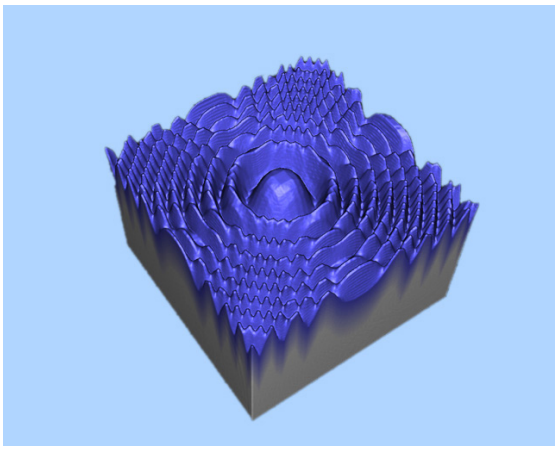
5.2 Progressive Cutting Tool

One important procedure for many surgical planning applications is the virtual bone cutting, especially in the context of orthopedic and maxillo-facial surgery [KGKG98]. In this work, advantage is taken of missing connectivity information and the resulting flexibility of point-based representations in order to design a simple as well as efficient cutting algorithm.

In this application example, signed distance fields (SDFs) have been used for the representation of cutting instruments [Har96]. It is trivial to define simple objects, like spheres and planes, with SDFs. More complex tools can be modeled using a combination of basic shapes. The cutting procedure is based on a point-wise evaluation of the SDF. The sign of the result value is used to determine the "in/out relation" for each primitive. Values that are smaller than the point size indicate elements that are located near the "border". Points, which are identified as "out", are assigned with a predefined material value offset. These offsets can be used to skip the associated points during rendering, e.g. by selecting low alpha values via the transfer function. Such an approach requires no copying of point information and memory reallocations, effecting in a high throughput. Primitives, which are marked as cut, can be removed explicitly after the completion of the interactive operations, including a postponed update of the underlying point data structure.

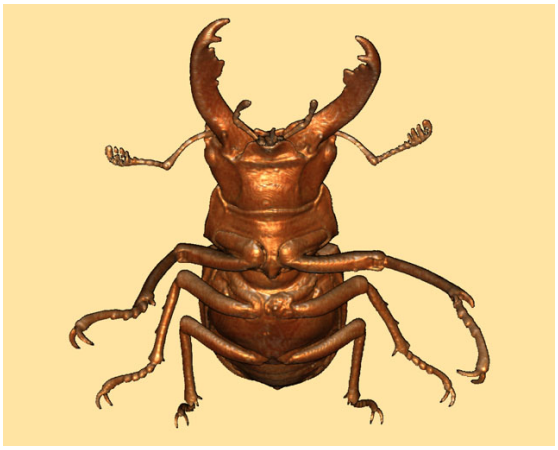


(a) Close-up view of colored RP Model



(b) Marschner-Lobb (10× supersampled)

(c) Stanford bunny



(d) TU Vienna Stagbeetle

Figure 5.5: Point-based renderings of non-medical datasets.

Basic Approach

Let $S : \Re^3 \rightarrow \Re$ represent the SDF of a cutting tool and O_i a point object with $1 \leq i \leq N$. Moreover, let $P_i = \{p_{i,1}, \ldots, p_{i,M_i}\} \subset \Re^3$ be a 3D point set and Ω_i a set of possible materials for O_i. Additionally, let $m_{i,j} \in \Omega_i$ be the material identifier that is associated with point $p_{i,j} \in P_i$. Let $c_i \in \Omega_i$ be the aforementioned material offset. It must be set to a value that is larger than any existing material identifier of the associated point object, i.e. $\forall 1 \leq i \leq N, \forall 1 \leq j \leq M_i : c_i > m_{i,j}$. After the operation, points with $m_{i,j} \geq c_i$ can be identified as cut. Algorithm 8 illustrates the basic procedure for point-based cutting.

If the cut part is detached from the original point set, all points with $m_{i,j} \geq c_i$ are deleted. Alternatively, the cut primitives can be copied to an own object instance, combined with a point-wise subtraction of c_i for all corresponding material identifiers. A sample cutting result is illustrated in figure 5.6.

Algorithm 8: Basic cutting procedure for point-based models.

```
1 for 1 ≤ i ≤ N do
2 |   for 1 ≤ j ≤ M_i do
3 |   |   if S(p_{i,j}) < 0 ∧ m_{i,j} < c_i then
4 |   |   |   m_{i,j} ← m_{i,j} + c_i;
5 |   |   end
6 |   end
7 end
```

Progressive Approach

The flexibility of point-based models does not always guarantee interactivity of the basic procedure. The cutting operation may require several seconds of processing time for complex cutting tools, high-resolution datasets and especially volumetric structures. A *progressive cutting* approach has been developed, inspired by the progressive refinement framework described in section 4.1.

Cutting requests are scheduled in a *first-in-first-out* (FIFO) queue on each new user input in order to assure interactivity. The point-wise cutting operations are executed whenever the application gets idle. They are interrupted when a user-defined time threshold is exceeded. Each element of the queue is associated with information about the operation state at cutting time, including the type and location of the cutting tool, involved point objects and the progress of the cutting procedure.

Let Q be the FIFO queue including all scheduled cutting requests, represented by the tuples $q = (S_q, T_{q,1}, \ldots, T_{q,N}, i_q, j_q) \in Q$. Each q contains the SDF S_q and the object transformation matrices $T_{q,i} \in \Re^4 \times \Re^4$ at cutting time. Additionally, it includes the indices i_q and j_q of the lastly processed object and corresponding point element. Both are used to control the cutting progress. Moreover, let $T_i \in \Re^4 \times \Re^4$ be the current transformation

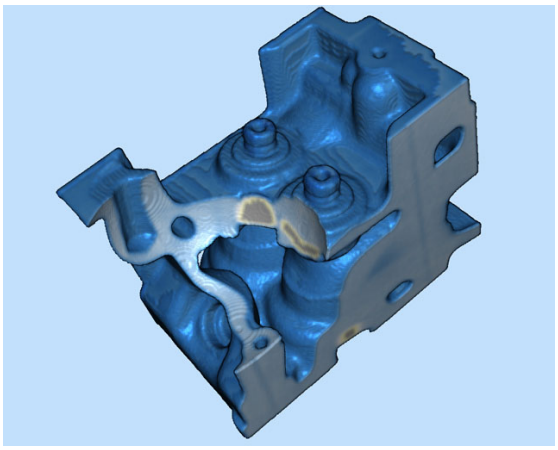


Figure 5.6: Cutting of a point-based dataset with a spherical tool.

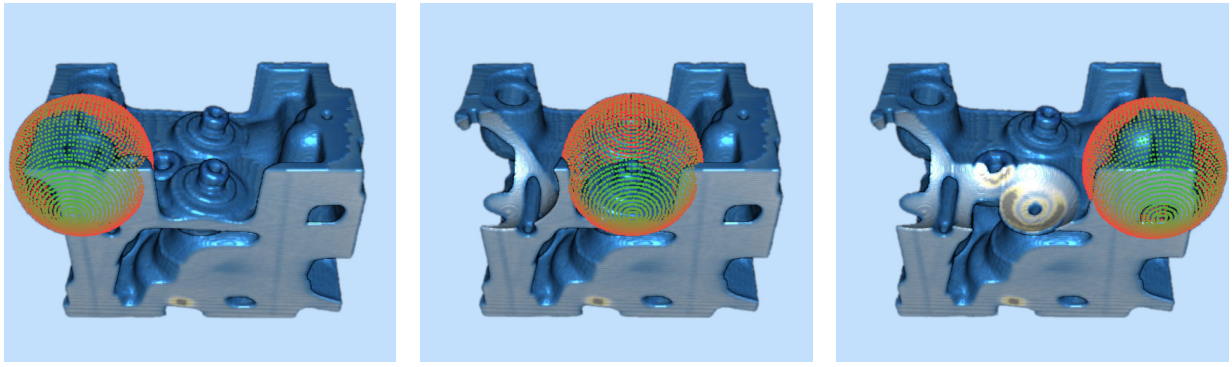


Figure 5.7: Illustration of the delay during progressive cutting. The red and green sphere represents the interactive cutting tool.

matrix for the i-th object. Finally, let $t^*, t_{cut} \in \Re$ be the current and target processing times, respectively. The progressive version of the basic cutting procedure is presented in algorithm 9.

Algorithm 9: Progressive cutting procedure for point-based models.

```
1  foreach q ∈ Q do
2  |  for i_q ≤ i ≤ N do
3  |  |  S'_q ← S_q · T_{q,i} · T_i^{-1};
4  |  |  for j_q ≤ j ≤ M_i do
5  |  |  |  if t* ≥ t_cut then
6  |  |  |  |  (i_q, j_q) ← (i, j);
7  |  |  |  |  Return;
8  |  |  |  end
9  |  |  |  if S'_q(p_{i,j}) < 0 ∧ m_{i,j} < c_i then
10 |  |  |  |  m_{i,j} ← m_{i,j} + c_i;
11 |  |  |  end
12 |  |  end
13 |  |  j_q ← 1;
14 |  end
15 |  Remove q ∈ Q;
16 end
```

Initially, indices i_q and j_q are initialized to the minimal index values. The transformation matrices $T_{q,i}$ and T_i are used to transform the SDF based on the location of the corresponding object at its cutting request time. Such an approach is more efficient than an element-wise transformation of the points. Figure 5.7 shows a processing sequence during progressive cutting.

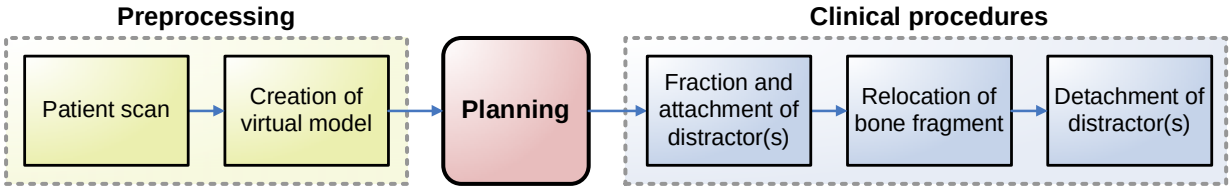


Figure 5.8: Overview of the distraction osteogenesis procedure.

5.3 Mandibular Distraction Osteogenesis

The final application example has been developed in the context of a clinical project. It comprises an interactive and virtual planning system for maxillo-facial-surgery. Its implementation has been facilitated by reusing many components from the PointPack library, point-based viewer application and progressive cutting tool.

Distraction osteogenesis is a surgical procedure for the treatment of skeletal deformities, created by a genetic defect, disease or accident. Usually, it consists of a surgically created fracture, followed by a gradually controlled displacement of the resulting bone fractures after a latency time of several days [YFH+04]. Slow relocations of the bone parts, typically 1 *mm* per day, invoke the stimulation of the human regeneration system and effect in a reconstruction of the missing soft tissue as well as bone structure. The fundament for all modern procedures is the work of the Russian orthopedic surgeon Ilizarov from the 1950s [Ili90]. Ilizarov has developed a medical apparatus, called *distractor*. It is mounted at the separated bone parts directly after surgical fracture and used to lengthen limb bones. After the arrival of the bone fracture at its desired end location and a bone consolidation phase of a few weeks, it is removed in a second surgical operation. The whole procedure is illustrated in figure 5.8.

The original distractor has been constructed from bicycle parts. Today's devices are highly complicated surgical instruments (figure 5.9)[1]. They invoke a step-wise displacement of bone parts along a three-dimensional path, usually including translations and rotations with respect to biomechanical constraints. This can result in an extensive parameter set with many degrees of freedom, especially for complex bone rearrangements in mandibular distraction osteogenesis. Therefore, sophisticated planning systems are required in order to support the medical expert during surgical planning.

A Point-Based Planning System

A computer-assisted planning procedure makes it possible to perform the whole operation virtually in advance. It requires the utilization of interactive visualization techniques and modifications of the underlying virtual patient model. Most existing planning systems are based on polygonal data structures [EST+00, MMM+06]. The goal in this work has been the exploitation of the potential that is provided by the point data and the evaluation of its

[1]Figure 5.9 is courtesy of Stryker cooperation, USA (`http://www.stryker.com`)

Figure 5.9: Interoral distractor device (Vazquez and Diner, Stryker cooperation).

relevance for clinical practice. The complete medical workflow has been implemented using point-based techniques, extending the visualization pipeline from chapter 3 [WJM+06]. After data import, generation and exploration of the resulting point-based isosurface model, the following workflow steps are executed:

1. **Osteotomy** - The first step consists of the simulation of the surgical fraction using the cutting procedure described in the previous section. A SDF-based flat box is used as the cutting tool. Its thickness can be used to imitate the loss of bone material, occurring during a real procedure (Figure 5.10(a)).

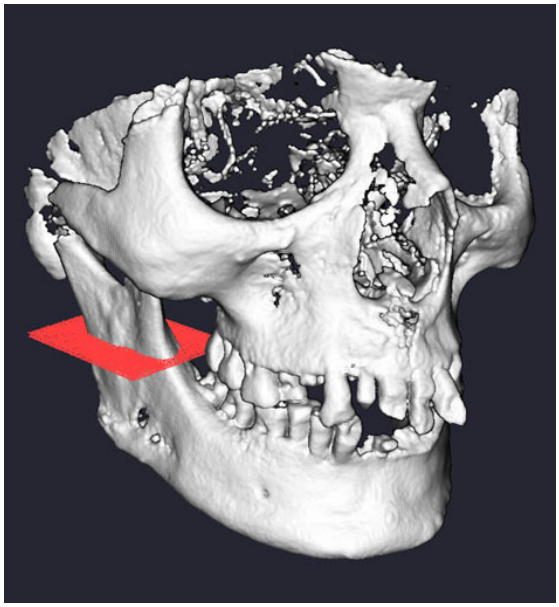
2. **Segmentation** - The distal bone fragment is identified using segmentation techniques. This step includes the execution of a connectivity filter on the point cloud, similar to an image-based "flood fill". Missing connectivity information is substituted by a point neighborhood relation (Figure 5.10(b)).

3. **Rearrangement** - The next goal is the geometric transformation of the segmented bone fragment to the desired end location that is defined manually by the medical expert. Interaction can be supported by a mid-sagittal plane that is used restrict the possible range of translations and rotations. (Figure 5.10(c)).

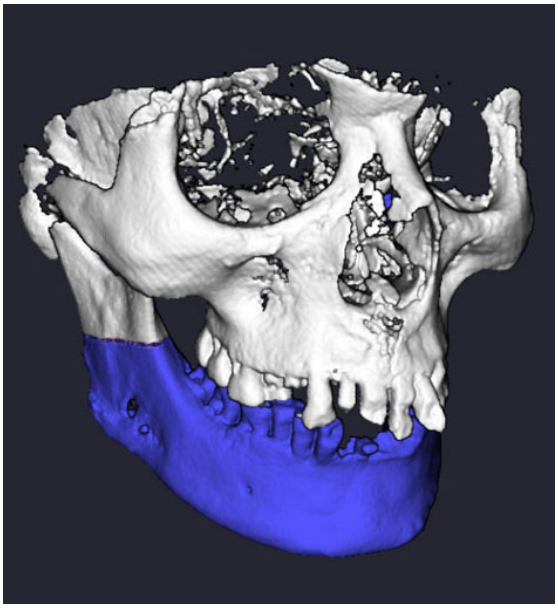
4. **Simulation** - Finally, the distraction path is computed based on a virtual movement of the distal fragment from the start to the previously defined end location. Calculations are constrained by a step-wise verification of biomechanical properties with respect to a minimal and maximal advancement (Figure 5.10(d)).

The output of the last workflow step is the *planning protocol*, including all information that is required for an iterative adjustment of real distractor parameters. Examples are the translation and rotation step sizes for the distractor device.

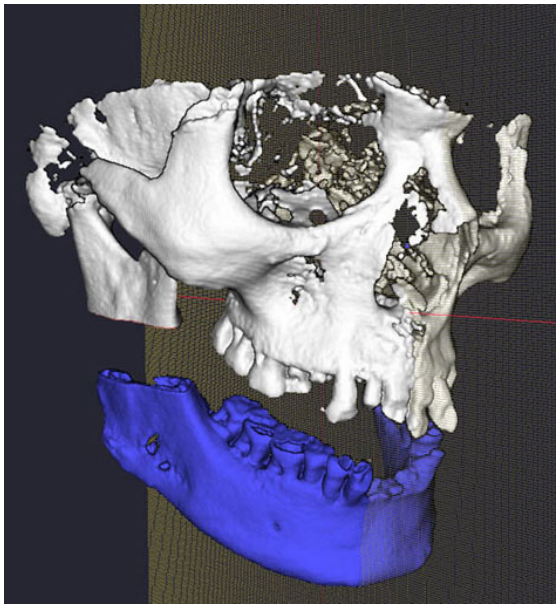
First experiments have confirmed the performance and memory advantages of the point-based approach. The osteotomy and rearrangement steps can be performed interactively due to the fast visualization and progressive cutting, even for complex isosurface models that have been generated from high resolution volumes with 1024^3 samples. The simulation computations require a few seconds, introducing only a negligible delay in the workflow.



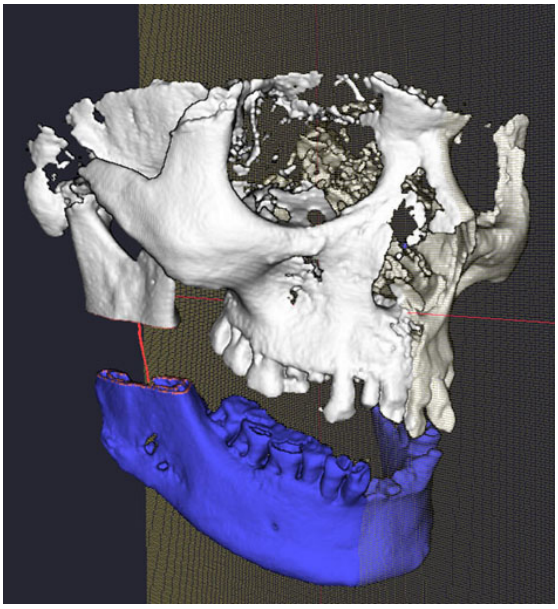
(a) Osteotomy



(b) Segmentation



(c) Rearrangement



(d) Simulation

Figure 5.10: Workflow of the mandibular distraction osteogenesis planning system.

The major performance bottleneck is the segmentation step. It requires several minutes for a model, generated from a 512^3 volume, and up to half an hour for higher resolution datasets. A traditional volume-based solution, e.g. 3D region growing, would achieve significantly better timing results by taking advantage of the implicit connectivity information. It might be more advantageous to consider the following alternative solution for such ap-

plications: Given the structured points data structures, described in section 3.1.1, the 3D model could be converted back to a volume dataset by simple "splatting" of the included point data into the volume. The next step would be the execution of a fast segmentation procedure, utilizing the connectivity of the volume grid. Then, the point model could be recreated from the segmented volume based on the generation algorithms, presented in section 3.1.2. Finally, the point-based workflow could be resumed. Indeed, this solution would break with the universal modeling approach, but it would make the usage of a more efficient segmentation possible.

The gathered practical experiences with the planning application have revealed that the utilization of simple and symmetric point primitives as well as a universal and sequential data representation have the potential to facilitate the development in visualization-related software projects. Nevertheless, they indicate also that there are application cases where traditional solutions provide better results. Therefore, point-based techniques should be considered complementary. More details and additional information can be found in Winter's diploma thesis [Win06].

Chapter 6

Conclusions

In the last chapter, the main aspects of this work will be recapitulated, followed by a discussion of the presented results. This includes the summary of the major advantages and challenges regarding the design of a purely point-based visualization pipeline for medical applications. The work will be concluded with a set of ideas for further investigation.

6.1 Summary

In this work, point-based techniques for interactive visualization in the context of medical applications have been investigated. The main challenges during the design of a typical medical processing pipeline have been addressed with regard to growing dataset sizes and graphics-hardware-oriented rendering. The following topics have been examined:

Point-Based Exploration - The major building blocks of a typical interactive application in computer-assisted medicine have been presented. In this context, the main principles for the design of point-based data structures have been defined. The resulting *span-triangle* provides fast data access and makes the tuning of exploration parameters in real-time possible. It can be converted to a compact operation data structure for further application-specific work steps. The linear data layout and abstract iterator interfaces establish a tight coupling between the point representation and efficient splatting-based rendering.

Universal Rendering - The major principles for the design of interactive renderers in the context of medical visualization have been defined. The *deferred splat shading* technique reduces the number of rendering passes to *1+1* and makes it possible to design a universal rendering component for surfaces and volumes as well as many other representations that are common in the medical domain. It has been shown how to generate visualizations at adequate quality from a highly irregular framebuffer content without "fuzzy splats". The presented point-based volume renderer is based on maximum-intensity-projection (MIP), including extensions like the simple hybrid

surface/MIP technique, DMIP and STS-MIP. Additionally, a splat-adaptive normal estimation technique has been integrated, called *micronormals averaging*. It supports the reduction of discontinuity artifacts that are common for z-buffer-based techniques.

Progressive Refinement Framework - An adaptive approach for progressive refinement of point-based models has been developed. Different portions of the dataset are associated with individual refinement priority values. The point density is adapted during rendering to ensure interactivity. Important regions of the virtual model are visualized at high resolution, similar to a "variable level-of-detail" (LOD) approach and β-acceleration in ray tracing. Inspired by Funkhouser's et al. adaptive display algorithm, examples for the definition of the cost and benefit functions for progressive refinement of point data have been presented. An extension of the aforementioned universal renderer has been provided, including smooth update during interaction.

Sequential Decomposition - A purely sequential LOD technique for point-based models has been presented, including the integration into the aforementioned progressive refinement framework. The LOD generation is interpreted as a reordering of the original rendering primitives. In this context, the *deflating spheres algorithm* for the computation of a sequential decomposition has been proposed. Alternatively, fast solutions can be generated by utilizing an approximate algorithm based on binary space partitioning. Moreover, the hole-free splatting of points with adaptive point sizes has been discussed, including the handling of feedback effects during progressive refinement.

Software Architecture and Examples - All developed point techniques have been integrated into a component-based software library, called *PointPack*. An overview of the main component architectures for point-based data processing and visualization has been given. Furthermore, three examples based on the PointPack library have been presented, including a viewer application for visualization of complex volumetric datasets, a progressive cutting tool for simulation of surgical fractions and an application prototype for the planning of mandibular distraction osteogenesis in maxillo-facial surgery.

6.2 Discussion

Advantages

First, advantage has been taken of the typical benefits of point-based representations in the context of interactive medical visualization. Missing connectivity information makes it possible to design simple, memory-efficient and flexible data structures as well as rendering algorithms. All presented point models are based on a compact point format - requiring only positional and material information - and can be therefore considered as minimal for the medical domain. Normal vectors are estimated on-the-fly from the framebuffer. Such

an approach is suitable for dynamic applications, minimizing "maintenance operations" after modifications of the data (e.g. surgical cuts, soft-tissue simulation).

Moreover, disconnected point elements facilitate the development of cache-efficient data structures and sequential algorithms. Primitives can be arranged in linear blocks of memory, so-called "point arrays", and processed independently in a stream-based fashion. This effects in hardware-efficiency and simple memory management. One example is the point-based exploration of medical volumes, as mentioned in section 3.1. It can be utilized for interactive isosurfacing, providing an explicit geometric representation for each isovalue without significant data-processing latency.

Furthermore, points have successfully been used to represent volumetric objects. In this context, the utilization of *extended surfaces* has proven to be a useful compromise between "thin" surfaces and "full" volumetric representations. They consist of a sequence of isosurfaces, associated with material information, and be generated efficiently by using the algorithmic components of the aforementioned point-based exploration infrastructure. Although "full" volumes can be modeled more efficiently with traditional voxels, point data is an attractive alternative for irregularly sampled models.

It has been shown in section 3.2 that using points makes it possible to mimic the most common visualizations from the medical area of application. This universality has allowed to design one single rendering component for different data and visualization types, including slices, surfaces and volumes. Exploitation of programmable graphics hardware and per-splat-exact lighting effect in high rendering rates at adequate visual quality. The provided results underline the advantage of point-based rendering over traditional approaches for irregular and complex models.

Additionally, the simplicity of the point primitive facilitates the development of progressive refinement and LOD techniques. As presented in chapter 4, lower detail levels can be generated just by a reduction of the primitive count for each point array and therewith point density. Therefore, one can take advantage of a fully sequential data layout without the build up and maintenance of hierarchical data structures.

The presented advantages of the point-based approach result in shorter processing times and a higher degree of interactivity for many medical applications. The aforementioned simplicity has the potential to facilitate development of visualization components and support rapid application development in the context of medical software projects.

Limitations

Nevertheless, missing connectivity information and the resulting high flexibility may turn into a problem for some areas of application. Algorithms that are based on frequent point neighborhood queries may cause a significant reduction of performance. One could take advantage of spatial acceleration data structures, but such an approach would usually be associated with the utilization of hierarchical information and therewith more complexity during design and development of an application. Typical examples are medical segmentation algorithms, data up-sampling and compression techniques, which exploit geometric coherence.

A symmetric point primitive with constant size per object (in model coordinates) has been assumed throughout the work. On the one hand, this strategy facilitates many processing and rendering tasks. On the other hand, it may result in unnecessarily high sampling densities and therewith memory consumption. For example, flat surface regions and anisotropic volume data could be represented by fewer "traditional" primitives, like triangles and voxels, respectively. In general, it is suggested to utilize point-based representations primarily for modeling of objects with complex geometry and a high number of primitives. This is true for most three-dimensional medical datasets that represent natural structures. For the representation of artificial objects, like CAD-based surgical tools, it might be more efficient to choose one of the traditional solutions.

Another issue is the degradation of visual quality for low-resolution models and in cases of high magnification. The reason is the utilization of a simple splat shape, driven by the decision to favor high performance over high rendering quality. Although, the visual appearance of the generated visualizations is adequate for noisy scanner data and for the display of a "big picture" of the medical scene, smooth objects may reveal aliasing artifacts that are noticeable and probably deranging for an experienced viewer.

Following Gross' conclusion, it should be noticed that points are not a replacement of polygonal meshes and voxel-based datasets for all possible kinds of medical visualizations [Gro06]. Points have the potential to complement traditional approaches for three-dimensional applications, which require a high level of interactivity and dynamic data modifications.

6.3 Future Work

In the following, possible extensions to the developed point-based techniques are presented, including some propositions for future research directions:

Smart out-of-core processing - The current data-processing pipeline is based on the following out-of-core strategy: If the allocation of a large memory chunk fails, it is subdivided into smaller blocks and distributed on different memory levels, including graphics, system and disk memory [vRLK05]. Such a general approach may result in significant performance degradation during the generation of the span-triangle for huge point sets with a wide exploration range. Preprocessing may be slowed down by memory swapping operations, invoked by the operating system. Possible solutions to this problem could be found by investigating advanced out-of-core techniques for data-processing. Another example is the utilization of "external memory" techniques for point-based progressive refinement: Point buckets with higher refinement priority could be transferred to higher memory levels in order to improve rendering performance during interaction.

Further performance optimizations - One problem of the presented rendering approach is the massive overdraw due to overlapping and hidden point primitives. Especially visualization of complex structures with volumetric information may be

associated with an overall performance degradation. Future work could include the investigation of backface culling and higher order hidden-surface-removal techniques for sequential point data structures. Moreover, the developed rendering iterators have been designed originally for exterior viewing of spacious three-dimensional structures. The current implementation may result in suboptimal traversal orders and point density allocations during progressive refinement for visualizations with enabled cut planes (e.g. slice and STS-MIP views) as well as fly-throughs (e.g. during virtual endoscopy). This problem could be solved by extending the proposed benefit functions. Additionally, during the development of the PointPack library the concentration has originally been on well-defined interfaces and clearly arranged source code. Alternatively, further speed improvements may be expected via low-level code optimizations.

Improved rendering quality - Further investigations could address also the problem of reduced visual quality for low resolution datasets and close-up views. A promising approach is dynamic up-sampling of point elements in regions of low screen-space sampling density. It could be integrated into the progressive refinement framework, for example. Another issue is the improvement of the presented z-buffer gradient estimation technique. Extensions could include the consideration of more image samples per normal operator and more sophisticated discontinuity checks.

Advanced volume rendering - The current rendering component supports only order-independent volume rendering, based on MIP. Some medical applications may also require composite rendering, including view-dependent alpha blending of volume samples. Therefore, further work could include the development of data structures for fast estimation of the point rendering order based on the current viewing parameters. Moreover, the integration of advanced non-photo-realistic and focus-in-context volume rendering techniques appears promising in the context of point-based medical visualization.

Enhanced adaptive display - The presented calculation of the visibility-based refinement order for adaptive display of point data is based on a slow *cone tracing* algorithm. The performance could be improved by incorporating a fast raycasting algorithm for the underlying point grid. Moreover, the current depth- and visibility-based refinement strategies provide satisfactory results only for opaque surfaces. It would be more appropriate to consider intensity-based arrangements of points for transparent and MIP-based visualizations. Another issue for further work is the extension of the adaptive display control. It could be also used for the shading step, e.g. to estimate the optimal viewport scaling factor during refinement.

Error-controlled level-of-detail - The estimation of the point density for the progressive refinement framework is dependent only on the actual target rendering time. Furthermore, one could incorporate also a screen space error metric for LOD control. Further objects would be rendered at lower resolutions, effecting in an improvement

of the overall performance. On the one hand, this issue has been considered of secondary importance in this work, because medical scenes have typically a low depth complexity. On the other hand, the presented point-based LOD technique based on sequential decomposition could be investigated in the context of non-medical application areas.

Optimized sequential decomposition - The quality of the sequential decomposition generation could be improved by computing point clusters during preprocessing. Calculations would be performed for each point cluster separately by considering only "connected" primitives. Then, smaller point sizes, especially for lower detail levels, could be expected. Additionally, only the approximate algorithm - based on binary space partitioning - provides reasonable timing results in practice for large datasets. Further optimizations of the deflating spheres algorithm or alternative approaches could be considered in future. Finally, the utilization of sequential decomposition for approximated collision detection computations is another promising domain for investigation.

In the future, it is planned to integrate the PointPack library in more clinical applications. One example is the utilization of the span-triangle for "time-based exploration", e.g. for the modeling and animation of a beating heart and knee joint dynamics for cardiac and orthopedic surgery, respectively. Other examples are motivated by the flexible and irregular modeling capabilities of the presented 3D representation, including particle-based simulation and modeling of tissue growth processes. Finally, it is considered to utilize the developed techniques in the context of commercial projects.

Bibliography

[AA03] A. Adamson and M. Alexa. Ray tracing point set surfaces. In *Shape Modeling International*, pages 272–282, 299, 2003.

[ABCO+01] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *Proceedings of IEEE Visualization*, 2001.

[AL99] D. G. Aliaga and A. Lastra. Automatic image placement to provide a guaranteed frame rate. In *SIGGRAPH Conference Proceedings*, pages 307–316, August 1999.

[Ama84] J. Amanatides. Ray tracing with cones. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 18, pages 129–135, July 1984.

[AUH+98] J. E. Anderson, C. Umans, M. Halle, P. Golland, M. Jakab, R. W. McCarley, F. A. Jolesz, M. E. Shenton, and R. Kikinis. Anatomy browser: Java-based interactive teaching tool for learning human neuroanatomy. *RSNA Electronic Journal*, 2, 1998.

[BAKW03] D. Baraff, J. Anderson, M. Kass, and A. Witkin. Physically-based modelling. In *SIGGRAPH Course Notes*, 2003.

[BC03] J. A. Bærentzen and N. J. Christensen. Hardware accelerated point rendering of isosurfaces. In *WSCG Conference Proceedings*, volume 11, February 2003.

[BDH+04] D. Bartz, H. Delingette, M. Hauth, A. Linney, N. Magnenat-Thalmann, K. Mueller, and Y. Wu. Advanced virtual medicine: Techniques and applications for medicine-oriented computer graphics. Eurographics Tutorial 6, 2004.

[BFGS86] L. Bergman, H. Fuchs, E. Grant, and S. Spach. Image rendering by adaptive refinement. *Computer Graphics (SIGGRAPH Conference Proceedings)*, 20(4):29–37, August 1986.

[BHZK05] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on todays GPUs. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 17–24, 2005.

[BK03] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern GPUs. In *Pacific Conference on Computer Graphics and Applications*, page 335, 2003.

[BKM00] J. Beutel, H. L. Kundel, and R. L. V. Metter, editors. *Handbook of Medical Imaging - Volume 1. Physics and Psychophysics*. SPIE Press, 2000.

[BPS96] C. L. Bajaj, V. Pascucci, and D. Schikore. Fast isocontouring for improved interactivity. In *IEEE Symposium on Volume Visualization*, page 39 pp., 1996.

[BSK04] M. Botsch, M. Spernat, and L. Kobbelt. Phong splatting. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 25–32, 2004.

[BWK02] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 53–64, June 26–28 2002.

[Car96] J. Carr. *Surface Reconstruction in 3D Medical Imaging*. PhD thesis, University of Canterbury, Christchurch, New Zealand, 1996.

[CCC87] R. L. Cook, L. Carpenter, and E. Catmull. The reyes image rendering architecture. *Computer Graphics (SIGGRAPH Conference Proceedings)*, 21(4):95–102, July 1987.

[CCM04] K. Cleary, H. Y. Chung, and S. K. Mun. OR2020 workshop overview: Operating room of the future. In *Proceedings of the International Congress on Computer Assisted Radiology and Surgery (CARS)*, volume 1268, pages 847–852, 2004.

[CGPD98] J. C. Carr, A. H. Gee1, R. W. Prager, and K. J. Dalton. Quantitative visualisation of surfaces from volumetric data. In *WSCG Conference Proceedings*, volume V. Skala, 1998.

[CH02] L. Coconu and H.-C. Hege. Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 43–52, June 26–28 2002.

[CHH+03] C. S. Co, B. Heckel, H. Hagen, B. Hamann, and K. I. Joy. Hierarchical clustering for unstructured volumetric scalar fields. In *Proceedings of IEEE Visualization*, pages 325–332, October 19–24 2003.

[CHJ03] C. S. Co, B. Hamann, and K. I. Joy. Iso-splatting: A point-based alternative to isosurface visualization. In *Pacific Conference on Computer Graphics and Applications*, pages 325–334, 2003.

[CHP+79] C. Csuri, R. Hackathorn, R. Parent, W. Carlson, and M. Howard. Towards an interactive high visual complexity animation system. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 13, pages 289–299, August 1979.

[CLL+88] H. E. Cline, W. E. Lorensen, S. Ludke, C. R. Crawford, and B. Teeter. Two algorithms for the reconstruction of surfaces from tomograms. *Medical Physics*, 15:320–327, 1988.

[CM93] R. A. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proceedings of IEEE Visualization*, pages 261–267, October 1993.

[CMM+97] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, April 1997.

[CN94] T. J. Cullip and U. Neumann. Accelerating volume reconstruction with 3d texture hardware. Technical report, University of North Carolina at Chapel Hill, 1994.

[CN01] B. Chen and M. X. Nguyen. POP: a hybrid point and polygon rendering system for large data. In *Proceedings of IEEE Visualization*, pages 45–52, October 21–26 2001.

[CRZP04] W. Chen, L. Ren, M. Zwicker, and H. Pfister. Hardware-accelerated adaptive EWA volume splatting. In *Proceedings of IEEE Visualization*, pages 67–74, 2004.

[Csé04] B. Csébfalvi. Interactive transfer function control for monte carlo volume rendering. In *IEEE Symposium on Volume Visualization*, pages 33–38, 2004.

[CSK03] B. Csébfalvi and L. Szirmay-Kalos. Monte carlo volume rendering. In *Proceedings of IEEE Visualization*, pages 449–456, October 2003.

[CT81] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *Computer Graphics (SIGGRAPH Conference Proceedings)*, 15(3):307–316, August 1981.

[CVCK03] J. Chhugani, S. Vishwanath, J. Cohen, and S. Kuma. ISOSLIDER: A system for interactive exploration of isosurfaces. In *Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*, pages 259–266, May 2003.

[DCSD02] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *Proceedings of IEEE Visualization*, pages 219–226, October 27– November 1 2002.

[Dee95] M. Deering. Geometry compression. In *SIGGRAPH Conference Proceedings*, pages 13–20, 1995.

[DG95] M. Desbrun and M. Gascuel. Animating soft substances with implicit surfaces. In *SIGGRAPH Conference Proceedings*, pages 287–290, August 1995.

[DG96] M. Desbrun and M.-P. Gascuel. Smoothed particles : A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation*, pages 61–76, 1996.

[DH92] J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Proceedings of the Workshop on Volume Visualization*, pages 91–98, October 1992.

[DLS05] T. K. Dey, G. Li, and J. Sun. Normal estimation for point clouds: A comparison study for a voronoi based method. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 39–46, 2005.

[DVS03] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. In *ACM Transactions on Graphics (SIGGRAPH)*, volume 22(3), pages 657–662, 2003.

[DWS+88] M. F. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt. The triangle processor and normal vector shader: A VLSI system for high performance graphics. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 22, pages 21–30, August 1988.

[ELPZ97] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.

[EST+00] P. Everett, E. B. Seldin, M. Troulis, L. B. Kaban, and R. Kikinis. A 3-D system for planning and simulating minimally-invasive distraction osteogenesis of the facial skeleton. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI*, volume 1935, pages 1029–1039, October 2000.

[Eve01] C. Everitt. Interactive order-independent transparency. NVIDIA OpenGL Applications Engineering, September 2001.

[EWE04] M. Eissele, D. Weiskopf, and T. Ertl. The G^2-buffer framework. In *Simulation und Visualisierung (SimVis)*, pages 287–298, March 2004.

[Fah06] M. Fahlén. Illumination for real-time rendering of large architectural environments. Linköping University, Sweden, Master Thesis, January 2006.

[FCOAS03] S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva. Progressive point set surfaces. *ACM Transactions on Graphics (SIGGRAPH)*, 22(4):997–1011, October 2003.

[FKP04] L. D. Floriani, L. Kobbelt, and E. Puppo. A survey on data structures for level-of-detail models. *Advances in Multiresolution for Geometric Modelling, Series in Mathematics and Visualization*, pages 49–74, 2004.

[FS93] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 27, pages 247–254, August 1993.

[Gal91] R. S. Gallagher. Span filtering: An efficient scheme for volume visualization of large finite element models. In *Proceedings of IEEE Visualization*, pages 68–75, October 1991.

[GBKG04a] S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. Memory efficient acceleration structures and techniques for CPU-based volume raycasting of large data. In *IEEE Symposium on Volume Visualization*, pages 1–8, 2004.

[GBKG04b] S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. VOTS: VOlume doTS as a point-based representation of volumetric data. *Computer Graphics Forum*, 23(3):661–668, 2004.

[GBP04a] G. Guennebaud, L. Barthe, and M. Paulin. Deferred splatting. *Computer Graphics Forum*, 23(3):653–660, septembre 2004.

[GBP04b] G. Guennebaud, L. Barthe, and M. Paulin. Real-time point cloud refinement. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 41–49, juin 2004.

[GBP05] G. Guennebaud, L. Barthe, and M. Paulin. Interpolatory refinement for real-time processing of point-based geometry. *Computer Graphics Forum*, 24(3):657–666, 2005.

[GBP06] G. Guennebaud, L. Barthe, and M. Paulin. Splat/mesh blending, perspective rasterization and tranparency for point-based rendering. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 49–57, 2006.

[GD98] J. P. Grossman and W. J. Dally. Point sample rendering. In *Eurographics Workshop on Rendering Techniques*, pages 181–192, June 1998.

[GM04] E. Gobbetti and F. Marton. Layered point clouds. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 113–120, 2004.

[GP03] G. Guennebaud and M. Paulin. Efficient screen space approach for hardware accelerated surfel rendering. In *Proceedings of the Conference on Vision, Modeling and Visualization (VMV)*, pages 485–494, November 19–21 2003.

[GPZ+04] M. Gross, H. Pfister, M. Zwicker, M. Alexa, M. Pauly, and M. Stamminger. Point-based computer graphics. SIGGRAPH Course Notes, 2004.

[GR85] D. Gordon and R. A. Reynolds. Image space shading of 3-dimensional objects. *Computer Vision, Graphics, and Image Processing*, 29:361–376, 1985.

[Gro98] M. H. Gross. Computer graphics in medicine: From visualization to surgery simulation. *Computer Graphics*, 32(1):53–56, February 1998.

[Gro99] M. H. Gross. Surgery simulation - a challenge for graphics and vision. In *Proceedings of the Workshop on Vision, Modeling and Visualization Workshop (VMV)*, November 1999.

[Gro06] M. Gross. Getting to the point...? *IEEE Computer Graphics and Applications*, 26(5):96–99, 2006.

[Har96] J. C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(9):527–545, 1996.

[Har04] S. Hargreaves. Deferred shading. Climax Corp., Game Developers Conference, 2004.

[HCSM00] J. Huang, R. Crawfis, N. Shareef, and K. Mueller. Fastsplats: optimized splatting on rectilinear grids. In *Proceedings of IEEE Visualization*, pages 219–226, 2000.

[HE03] M. Hopf and T. Ertl. Hierarchical splatting of scattered data. In *Proceedings of IEEE Visualization*, pages 433–440, 2003.

[HMBG01] H. Hauser, L. Mroz, G. I. Bischi, and M. E. Gröller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, July 2001.

[HMS95] W. Heidrich, M. McCool, and J. Stevens. Interactive maximum projection volume rendering. In *Proceedings of IEEE Visualization*, pages 11–18, 1995.

[Hop96] H. Hoppe. Progressive meshes. In *SIGGRAPH Conference Proceedings*, pages 99–108, 1996.

[Hou73] G. N. Hounsfield. Computerized transverse axial scanning (tomography). 1. description of system. *The British Journal of Radiology (BJR)*, 46(552):1016–1022, December 1973.

[HPP+95] K. H. Höhne, B. Pflesser, A. Pommertt, M. Riemer, T. Schiemann, R. Schubert, and U. Tiede. A new representation of knowledge concerning human anatomy and function. *Nature Medicine*, 1(6):506–511, 1995.

[HSS+05] M. Hadwiger, C. Sigg*, H. Scharsach, K. Buhler, and M. Gross*. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[HWG83] F. S. Hill, Jr., S. Walker, Jr., and F. Gao. Interactive image query system using progressive transmission. *Computer Graphics (SIGGRAPH Conference Proceedings)*, 17(3):323–330, July 1983.

[Ili90] G. A. Ilizarov. Clinical application of the tension-stress effect for limb lengthening. *Clinical Orthopaedics and Related Research*, 250:8–26, January 1990.

[JA03] B. Jezek and K. Antos. Improved gradient estimation methods for rendering of volume data. In *WSCG Conference Proceedings*, volume 11, February 2003.

[JDH+05] T. Jansen, T. Dreiseidler, N. Hanssen, L. Ritter, B. von Rymon-Lipinski, and E. Keeve. Julius a software framework for rapid application development in computer-aided surgery. In *Annual Conference of the German Society for Biomedical Engineering (BMT)*, September 2005.

[KB04] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, 2004.

[KCoSY99] A. Kadosh, D. Cohen-or, O. Shibolet, and R. Yagel. Rendering discrete surfaces from close-up views. Technical Report, Computer Science Department, School of Mathematical Sciences, Tel-Aviv University, Israel, November 14 1999.

[KD98] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium on Volume Visualization*, pages 79–86, 1998.

[KGKG98] E. Keeve, S. Girod, R. Kikinis, and B. Girod. Deformable modeling of facial tissue for craniofacial surgery simulation. *Computer Aided Surgery*, 3(5):228–238, 1998.

[KK04] H. Kawata and T. Kanai. Image-based point rendering for multiple range images. In *International Conference on Information Technology and Applications (ICITA)*, pages 478–483, January 2004.

[KK05] H. Kawata and T. Kanai. Direct point rendering on GPU. In *Proceedings of the International Symposium on Advances in Visual Computing (ISVC)*, volume 3804, pages 587–594, 2005.

[KLK04] J. Kim, S. Lee, and L. Kobbelt. View-dependent streaming of progressive meshes. In *Shape Modeling International*, pages 209–220, 2004.

[KLS96] R. Klein, G. Liebich, and W. Straßer. Mesh reduction with error control. In *Proceedings of IEEE Visualization*, pages 311–318, October 27–November 1 1996.

[KMH00] O. Kreylos, K.-L. Ma, and B. Hamann. A multi-resolution interactive previewer for volumetric data on arbitrary meshes. In *Proceedings of the Workshop on Computer Graphics and Virtual Reality*, December 2000.

[KP01] K. H. Kim and H. W. Park. A fast progressive method of maximum intensity projection. *Computerized Medical Imaging and Graphics*, 25(3):433–441, 2001.

[KSW04] P. Kipfer, M. Segal, and R. Westermann. Uberflow: A GPU-based particle engine. In *Graphics Hardware*, pages 115–122, 2004.

[KSW05] J. Krüger, J. Schneider, and R. Westermann. Compression and rendering of iso-surfaces and point sampled geometry. *The Visual Computer*, 2005.

[KSW06] J. Krüger, J. Schneider, and R. Westermann. ClearView: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization)*, 12(5), September-October 2006.

[KV01] A. Kalaiah and A. Varshney. Differential point rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 139–150, 2001.

[KV03] A. Kalaiah and A. Varshney. Statistical point geometry. In *Symposium on Geometry Processing*, pages 107–115, 2003.

[KW03] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization*, pages 287–292, 2003.

[KW05] P. Kipfer and R. Westermann. GPU construction and transparent rendering of iso-surfaces. In *Proceedings of the Conference on Vision, Modeling and Visualization (VMV)*, pages 241–248, 2005.

[LC87] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH Conference Proceedings*, pages 163–169, 1987.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[Lev90a] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[Lev90b] M. Levoy. Volume rendering by adaptive refinement. *The Visual Computer*, 6(1):2–7, 1990.

[LH91] D. Laur and P. Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Computer Graphics (SIGGRAPH Conference Proceedings)*, 25(4):285–288, 1991.

[LHJ99] E. C. LaMar, B. Hamann, and K. I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of IEEE Visualization*, pages 355–362, October 1999.

[Lin01] L. Linsen. Point cloud representation. Technical Report No. 2001-3, Fakultät fur Informatik, Universität Karlsruhe, Germany, March 01 2001.

[LKM01] E. Lindholm, M. J. Kligard, and H. Moreton. A user-programmable vertex engine. In *SIGGRAPH Conference Proceedings*, pages 149–158, 2001.

[LPR+00] M. Levoy, K. Pulli, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, B. Curless, J. Shade, and D. Fulk. The digital michelangelo project: 3D scanning of large statues. In *SIGGRAPH Conference Proceedings*, pages 131–144, July 23–28 2000.

[LSJ96] Y. Livnat, H.-W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[LT04] Y. Livnat and X. Tricoche. Interactive point-based isosurface extraction. In *Proceedings of IEEE Visualization*, pages 457–464, 2004.

[LW85] M. Levoy and T. Whitted. The use of points as a display primitive. Technical Report TR 85-022, University of North Carolina at Chapel Hill, 1985.

[LW90] M. Levoy and R. Whitaker. Gaze-directed volume rendering. *SIGGRAPH Conference Proceedings*, 24(2):217–223, 1990.

[MBAB04] P. N. Mallón, M. Bóo, M. Amor, and J. D. Bruguera. Algorithms and hardware for data compression in point rendering applications. In *WSCG Conference Proceedings (Short papers)*, pages 173–180, 2004.

[MC98] K. Mueller and R. Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. In *Proceedings of IEEE Visualization*, pages 239–246, October 1998.

[MD03] C. Moenning and N. A. Dodgson. A new point cloud simplification algorithm. In *IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP)*, sep 2003.

[MD04] C. Moenning and N. A. Dodgson. Intrinsic point cloud simplification. In *GraphiCon*, sep 2004.

[ME04] B. Mora and D. S. Ebert. Instant volumetric understanding with order-independent volume rendering. *Computer Graphics Forum*, 23(3):489–498, 2004.

[ME05] B. Mora and D. S. Ebert. Low-complexity maximum intensity projection. *ACM Transactions on Graphics*, 24(4):1392–1416, October 2005.

[MGK99] L. Mroz, E. Gröller, and A. König. Real-time maximum intensity projection. In *Data Visualization*, pages 135–144. Springer-Verlag Wien, May 1999.

[MHB+00] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. In *Proceedings of IEEE Visualization*, pages 81–90, October 2000.

[MLD88] M. Magnusson, R. Lenz, and P. E. Danielsson. Evaluation of methods for shaded surface display of CT-volumes. In *International Conference on Pattern Recognition*, pages 1287–1294, November 1988.

[MMC99] K. Mueller, T. Möller, and R. Crawfis. Splatting without the blur. In *Proceedings of IEEE Visualization*, pages 363–370, October 1999.

[MMM+06] M. Meehan, D. Morris, C. R. Maurer, A. K. Antony, F. Barbagli, K. Salisbury, and S. Girod. Virtual 3d planning and guidance of mandibular distraction osteogenesis. *Computer Aided Surgery*, 11(2):51–62, March 2006.

[Moe06] C. Moenning. Intrinsic point-based surface processing. Technical Report UCAM-CL-TR-658, University of Cambridge, Computer Laboratory, January 2006.

[MY96] K. Mueller and R. Yagel. Fast perspective volume rendering with splatting by utilizing a ray-driven approach. In *Proceedings of IEEE Visualization*, pages 65–72, October 27–November 1 1996.

[MZ03] F. Meng and H. Zha. Streaming transmission of point-sampled geometry based on view-dependent level-of-detail. In *International Conference on 3D Digital Imaging and Modeling (3DIM)*, pages 466–473, 2003.

[NBHJ03] C. Nuber, R. W. Bruckschen, B. Hamann, and K. I. Joy. Interactive visualization of very large medical datasets using point-based rendering. In *Proceedings of Medical Imaging (MI) - Visualization, Image-guided Procedures, and Display*, volume 5029, 2003.

[NCKG00] L. Neumann, B. Csébfalvi, A. König, and E. Gröller. Gradient estimation in volume data using 4D linear regression. *Computer Graphics Forum*, 19(3):351–358, August 2000.

[NM05] N. Neophytou and K. Mueller. GPU accelerated image aligned splatting. In *Joint Eurographics - IEEE VGTC Workshop on Volume Graphics*, pages 197–205, 2005.

[NMF99] L. P. Nedel, I. H. Manssour, and C. M. D. S. Freitas. Computer graphics and medicine. Tutorial of SIBGRAPI - International Symposium on Computer Graphics, Image Processing and Vision, 1999.

[NMM+06] N. Neophytou, K. Mueller, K. McDonnell, W. Hong, X. Guan, H. Qin, and A. Kaufman. GPU-accelerated volume splatting with elliptical RBFs. In *Joint Eurographics - IEEE TCVG Symposium on Visualization (EuroVis)*, May 2006.

[NRJ93] Napin, Rubin, and Jeffrey. STS-MIP: A new reconstruction technique for CT of the chest. *Journal of Computer Assisted Tomography*, 17(5):832–838, 1993.

[Paj03] R. Pajarola. Efficient level-of-details for point based rendering. In *Computer Graphics and Imaging*, pages 141–146, 2003.

[Paj05] R. Pajarola. Stream-processing points. In *Proceedings of IEEE Visualization*, page 31, 2005.

[PBCK05] B. Purnomo, J. Bilodeau, J. D. Cohen, and S. Kumar. Hardware-compatible vertex compression using quantization and simplifcation. In *Graphics Hardware*, pages 53–62, 2005.

[PBMH02] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *ACM Transactions on Graphics (SIGGRAPH)*, pages 703–712, 2002.

[PF04] F. Policarpo and F. Fonseca. Deferred shading tutorial. Brazilian Symposia on Games and Digital Entertainment (SBGAMES), 2004.

[PGK02] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of IEEE Visualization*, pages 163–170, October 27– November 1 2002.

[PHK+03] V. Pekar, D. Hempel, G. Kiefer, M. Busch, and J. Wees. Efficient visualization of large medical image datasets on standard PC hardware. In *Proceedings of the Symposium on Data Visualisation*, pages 135–140, 2003.

[Pho75] B.-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[PP03] B. Preim and H.-O. Peitgen. Smart 3d visualizations for clinical applications. In *Proceedings of the Smart Graphics Symposium*, pages 343–353, 2003.

[PSG03] R. Pajarola, M. Sainz, and P. Guidotti. Object-space blending and splatting of points. Technical Report UCI-ICS-03-01, The School of Information and Computer Science, University of California Irvine, April 15 2003.

[PSG04] R. Pajarola, M. Sainz, and P. Guidotti. Confetti: Object-space point blending and splatting. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):598–608, 2004.

[PSL05] R. Pajarola, M. Sainz, and R. Lario. XSplat: External memory multiresolution point visualization. In *IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP)*, pages 628–633, 2005.

[PZvBG00] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *SIGGRAPH Conference Proceedings*, pages 335–342, 2000.

[RBH+04] L. Ritter, Z. Burgielski, N. Hanssen, T. Jansen, M. Lievin, R. Sader, H.-F. Zeilhofer, and E. Keeve. 3d interactive segmentation of bone for computer-aided surgical planning. In *Proceedings of the Annual Conference of the International Society for Computer Assisted Orthopedic Surgery (CAOS)*, 2004.

[RE01] P. Rheingans and D. Ebert. Volume illustration: Nonphotorealistic rendering of volume models. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):253–264, 2001.

[Ree83] W. T. Reeves. Particle systems – A technique for modeling a class of fuzzy objects. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 17, pages 359–376, July 1983.

[Ree85] W. T. Reeves. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, page 313, 1985.

[Rev97] B. Revet. DICOM cook book for implementations in modalities. Philips Medical Systems, Document Number XPR080-970004.00, 1997.

[RL00] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH Conference Proceedings*, pages 343–352, 2000.

[RL01] S. Rusinkiewicz and M. Levoy. Streaming QSplat: a viewer for networked visualization of large, dense models. In *Symposium on Interactive 3D Graphics (SI3D)*, pages 63–68, 2001.

[RLB+01] J. Ribelles, A. Lopez, O. Belmonte, J. Remolar, and M. Chover. Variable resolution level-of-detail of multiresolution ordered meshes. In *WSCG Conference Proceedings*, 2001.

[RPZ02] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum*, 21(3):461–470, September 2002.

[SA06] M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 2.1). Silicon Graphics, Inc., 2006.

[SC01] N. Shareef and R. Crawfis. A view-dependent approach to MIP for very large data. In *Proceedings of SPIE Electronic Imaging (EI)*, volume 4665, pages 13–21, 2001.

[SD01] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 151–162, 2001.

[SDWE98] O. Sommer, A. Dietz, R. Westermann, and T. Ertl. An interactive visualization and navigation tool for medical volume data. In *International Conference in Central Europe on Computer Graphics and Visualization*, volume 2, pages 362–371, February 1998.

[Sed98] R. Sedgewick. *Algorithms in C++*. Series in Computer Science. Addison-Wesley, 3rd edition, 1998.

[SGS95] G. Sakas, M. Grimm, and A. Savopoulos. Optimized maximum intensity projection (MIP). In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 51–63, 1995.

[SGwHS98] J. W. Shade, S. J. Gortler, L. wei He, and R. Szeliski. Layered depth images. In *SIGGRAPH Conference Proceedings*, pages 231–242, July 19–24 1998.

[SHLJ96] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In *Proceedings of IEEE Visualization*, pages 287–294, 1996.

[SMM+97] J. E. Swan II, K. Mueller, T. Möller, N. Shareef, R. A. Crawfis, and R. Yagel. An anti-aliasing technique for splatting. In *Proceedings of IEEE Visualization*, pages 197–204, November 1997.

[SP04] M. Sainz and R. Pajarola. Point-based rendering techniques. *Computers & Graphics*, 28(6):869–879, 2004.

[SPL04] M. Sainz, R. Pajarola, and R. Lario. Points reloaded: Point-based rendering revisited. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 121–128, 2004.

[ST90] T. Saito and T. Takahashi. Comprehensible rendering of 3-D shapes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 24, pages 197–206, August 1990.

[THB+90] U. Tiede, K. H. Hoehne, M. Bomans, A. Pommert, M. Riemer, and G. Wiebecke. Investigation of medical 3D-rendering algorithms. *IEEE Computer Graphics and Applications*, 10(2):41–53, March 1990.

[TIP05] C. Tietjen, T. Isenberg, and B. Preim. Combining Silhouettes, Surface, and Volume Rendering for Surgery Education and Planning. In *Joint Eurographics - IEEE TCVG Symposium on Visualization (EuroVis)*, pages 303–310, 2005.

[Toe03] K. Toennies. Seeing and understanding 3-d medical images through 2-d renditions. In *Proceedings of the International Congress on Computer Assisted Radiology and Surgery (CARS)*, volume 1256, pages 1279–1284, 2003.

[VBB+06] F. P. Vidal, F. Bello, K. W. Brodlie, N. W. John, D. Gould, R. Phillips, and N. Avis. Principles and applications of computer graphics in medicine. *Computer Graphics Forum*, 25(1):113–137, March 2006.

[VGH+05] I. Viola, M. E. Gröller, M. Hadwiger, K. Bühler, B. Preim, M. C. Sousa, D. S. Ebert, and D. Stredney. Illustrative visualization. IEEE Visualization Tutorial 4, 2005.

[vRLHJ+04] B. von Rymon-Lipinski, N. Hanssen, T. Jansen, L. Ritter, and E. Keeve. Efficient point-based isosurface exploration using the span-triangle. In *Proceedings of IEEE Visualization*, pages 441–448, 2004.

[vRLK04] B. von Rymon-Lipinski and E. Keeve. European patent no. PCT-EP2005-052599, Methode und System zur interaktiven Anzeige und Exploration von internen Strukturen innerhalb eines Festkörpers unter Verwendung von punktbasierten Isooberflächen, 2004.

[vRLK05] B. von Rymon-Lipinski and E. Keeve. German patent no. 10 2005 032 349.9-53, Der Flussspeicher, Methode und System zur Repräsentation und Organisation großen Datensätzen, insbesondere für medizinische Visualisierungsapplikationen, 2005.

[Wan04] M. Wand. *Point-Based Multi-Resolution Rendering*. PhD thesis, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, 2004.

[WCJ05] K. W. Waters, C. S. Co, and K. I. Joy. Isosurface extraction using fixed-sized buckets. In *Joint Eurographics - IEEE VGTC Symposium on Visualization (EuroVis)*, pages 207–214, 2005.

[WE98] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH Conference Proceedings*, pages 169–177, 1998.

[Wes90] L. Westover. Footprint evaluation for volume rendering. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 24, pages 367–376, August 1990.

[WFP+01] M. Wand, M. Fischer, I. Peter, F. M. auf der Heide, and W. Straßer. The randomized z-buffer algorithm: interactive rendering of highly complex scenes. In *SIGGRAPH Conference Proceedings*, pages 361–370, 2001.

[WGE+04] M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray, and S. Würmlin. Progressive compression of point-sampled models. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 95–102, 2004.

[WGH83] S. Walker, Jr., F. Gao, and F. S. Hill, Jr. Interactive image query system using progressive transmission. In *Computer Graphics (SIGGRAPH Conference Proceedings)*, volume 17, pages 323–330, July 1983.

[WH94] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *SIGGRAPH Conference Proceedings*, pages 269–278, July 1994.

[Win06] A. Winter. An application for 3d planning of mandibular distraction osteogenesis utilizing point-based visualization. Diploma thesis, Private Fernfachhochschule Darmstadt, Fachbereich Informatik, October 2006.

[WJM+06] A. Winter, T. Jansen, R. A. Mischkowski, L. Ritter, B. von Rymon Lipinski, and E. Keeve. An application for 3d-planning of mandibular distraction osteogenesis utilizing point-based visualization. In *Annual Congress of the German Society for Computer- and Robot-Assisted Surgery (CURAC)*, October 2006.

[WK04] J. Wu and L. Kobbelt. Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum*, 23(3):643–652, 2004.

[WM03] T. Welsh and K. Mueller. A frequency-sensitive point hierarchy for images and volumes. In *Proceedings of IEEE Visualization*, pages 425–432, 2003.

[WMQR02] B. Wilson, K.-L. Ma, J. Qiang, and R. Ryne. Interactive visualization of particle beams for accelerator design. In *International Conference on Computational Science (ICCS)*, volume 2331, pages 352–361, April 2002.

[WS05] I. Wald and H.-P. Seidel. Interactive ray tracing of point-based models. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 9–16, 2005.

[WS06] M. Wimmer and C. Scheiblauer. Instant points. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, July 2006.

[WvG92] J. Wilhelms and A. van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[WW03] M. Wimmer and P. Wonka. Rendering time estimation for real-time rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 118–129, June 2003.

[WWH+00] M. Weiler, R. Westermann, C. D. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *IEEE Symposium on Volume Visualization*, pages 7–13, 2000.

[WZK05] J. Wu, Z. Zhang, and L. Kobbelt. Progressive splatting. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 25–32, 2005.

[XC03] D. Xue and R. Crawfis. Efficient splatting using modern graphics hardware. *Journal of Graphics Tools (JGT)*, 8(3):1–21, 2003.

[YC04] X. Yuan and B. Chen. Illustrating surfaces in volume. In *Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*, pages 9–16, 2004.

[YCK92] R. Yagel, D. Cohen, and A. E. Kaufman. Normal estimation in 3D discrete space. *The Visual Computer*, 8(5&6):278–291, 1992.

[YFH+04] J. C. Yu, J. Fearon, R. J. Havlik, S. R. Buchman, and J. W. Polley. Distraction osteogenesis of the craniofacial skeleton. *Plastic and Reconstructive Surgery*, 114(1):1E–20E, 2004.

[ZK06] H. Zhang and A. Kaufman. Interactive point-based isosurface exploration and high-quality rendering. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization)*, 12(5), September-October 2006.

[ZP06] Y. Zhang and R. Pajarola. Single-pass point rendering and transparent shading. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 37–48, 2006.

[ZPvBG01a] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. EWA volume splatting. In *Proceedings of IEEE Visualization*, pages 29–36, October 21–26 2001.

[ZPvBG01b] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *SIGGRAPH Conference Proceedings*, pages 371–378, 2001.

[ZPvBG02] M. Zwicker, H. Pfister, J. van Baar, and M. H. Gross. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.

[ZRB+04] M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly. Perspective accurate splatting. In *Proceedings of the Graphics Interface Conference*, pages 247–254, May 2004.