FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Sampled and Prefiltered Anti-Aliasing on Parallel Hardware

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

MSc. Thomas Auzinger
Matrikelnummer 0102176

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assoc. Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Diese Dissertation haben begutachtet:

Assoc. Prof. Ing. Jiří Bittner, PhD.	Assoc. Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Wien, 20. März 2015

Thomas Auzinger

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Sampled and Prefiltered Anti-Aliasing on Parallel Hardware

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

MSc. Thomas Auzinger
Registration Number 0102176

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Assoc. Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

The dissertation has been reviewed by:

Assoc. Prof. Ing. Jiří Bittner, PhD.	Assoc. Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer

Vienna, 20th March, 2015

Thomas Auzinger

Erklärung zur Verfassung der Arbeit

MSc. Thomas Auzinger
Neubau 1/2/6, 2440 Gramatneusiedl, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. März 2015

Thomas Auzinger

Abstract

A fundamental task in computer graphics is the generation of two-dimensional images. Prominent examples are the conversion of text or three-dimensional scenes to formats that can be presented on a raster display. Such a conversion process—often referred to as rasterization or sampling—underlies inherent limitations due to the nature of the output format. This causes not only a loss of information in the rasterization result, which manifests as reduced image sharpness, but also causes corruption of the retained information in form of aliasing artifacts. Commonly observed examples in the final image are staircase artifacts along object silhouettes or Moiré-like patterns.

The main focus of this thesis is on the effective removal of such artifacts—a process that is generally referred to as anti-aliasing. This is achieved by removing the offending input information in a filtering step during rasterization. In this thesis, we present different approaches that either minimize computational effort or emphasize output quality.

We follow the former objective in the context of an applied scenario from medical visualization. There, we support the investigation of the interiors of blood vessels in complex arrangements by allowing for unrestricted view orientation. Occlusions of overlapping blood vessels are minimized by automatically generating cut-aways with the help of an occlusion cost function. Furthermore, we allow for suitable extensions of the vessel cuts into the surrounding tissue. Utilizing a level of detail approach, these cuts are gradually smoothed with increasing distance from their respective vessels. Since interactive response is a strong requirement for a medical application, we employ fast sample-based anti-aliasing methods in the form of visibility sampling, shading supersampling, and post-process filtering.

We then take a step back and develop the theoretical foundations for anti-aliasing methods that follow the second objective of providing the highest degree of output quality. As the main contribution in this context, we present exact anti-aliasing in the form of prefiltering. By computing closed-form solutions of the filter convolution integrals in the continuous domain, we circumvent any issues that are caused by numerical integration and provide mathematically correct results. Together with a parallel hidden-surface elimination, which removes all occluded object parts when rasterizing three-dimensional scenes, we present a ground-truth solution for this setting with exact anti-aliasing. We allow for complex illumination models and perspective-correct shading by combining

visibility prefiltering with shading sampling and generate a ground-truth solution for multisampling anti-aliasing.

All our aforementioned methods exhibit highly parallel workloads. Throughout the thesis, we present their mapping to massively parallel hardware architectures in the form of graphics processing units. Since our approaches do not map to conventional graphics pipelines, we implement our approach using general-purpose computing concepts. This results in decreased runtime of our methods and makes all of them interactive.

Kurzfassung

Die Erzeugung zweidimensionaler Bilder zählt zu den grundlegenden Aufgaben der Computergrafik. Bedeutende Beispiele sind die Umwandlung von Text oder dreidimensionalen Szenen in ein Format, das sich zur Ausgabe auf einem Rasterdisplay eignet. Eine solche Konvertierung—oft Rasterisierung oder Sampling genannt—unterliegt inhärenten Limitierungen, die sich in der Natur des Ausgabeformates begründen. Dies bewirkt nicht nur einen Informationsverlust im Rasterisierungsresultat, der sich einer Reduktion der Bildschärfe äußert, sondern verursacht auch eine Verfälschung der verbleibendend Information in Form von Aliasingartefakten. Gängige Beispiele sind Treppenmuster an Objektsilhouetten oder Moiré-artige Muster.

Das Hauptaugenmerk der hier vorliegenden Dissertation ist die effiziente Behebung ebendieser Artefakte—ein Vorgang, der üblicherweise als Anti-Aliasing bezeichnet wird. Dies wird durch ein Entfernen der unzulässigen Eingangsinformation während der Rasterisierung mithilfe einer Filterung erreicht. Wir präsentieren in dieser Dissertation verschiedene Ansätze, die ihr Hauptaugenmerk entweder auf eine Minimierung des Rechenaufwandes oder auf maximale Ausgabequalität legen.

Wir folgen der ersten Zielsetzung in einer Anwendung aus der medizinischen Visualisierung, in der wir die Inspektion von Blutsgefäßinnenräumen von komplexe Gefäßanordnungen unterstützen, indem wir eine beliebige Betrachtungsrichtung ermöglichen. Verdeckungen von überlappenden Gefäßen werden durch das automatisierte Einfügen von Ausschnitten minimiert, wofür eine Verdeckungskostenfunktion eingesetzt wird. Darüber hinaus wird eine Erweiterung der Blutgefäßschnitte in das umliegende Gewebe ermöglicht. Mittels einem detailgradbasierendem Verfahren werden die Schnitte mit steigender Distanz von ihrem zugehörigen Gefäß graduell geglättet. Eine interaktive Reaktionszeit unserer Implementation wird durch eine Kombination von Sichtbarkeitssampling, Schattierungssupersampling und nachträglicher Filterung ermöglicht.

Im Weiteren nehmen wir einen anderen Ansatz und entwickeln die theoretischen Grundlagen für Anti-Aliasing Methoden, die der zweiten Zielsetzung eines Höchstgrades an Ausgabequalität folgen. Als Hauptbeitrag in diesem Zusammenhang präsentieren wir exaktes Anti-Aliasing in der Form von Präfilterung. Durch das Berechnen geschlossener Lösungen der auftretenden Filterungsfaltungen im Kontinuierlichen, umgehen wir sämtliche Problematiken, die sich durch numerische Integrationsansätze ergeben, und bieten mathematisch korrekte Resultate. Zusammen mit einer parallelen Elimination

versteckter Oberflächen, die alle verdeckten Objektteile bei der Rasterisierung dreidimensionaler Szenen entfernt, präsentieren wir eine Referenzlösung für dieses Szenario mittels exaktem Anti-Aliasing. Weiters ermöglichen wir die Verwendung von komplexen Beleuchtungsmodellen und perspektivisch korrekter Schattierung indem Sichtbarkeitspräfilterung mit Schattierungsampling kombiniert wird; woraus eine Referenzlösung für Multisampling-Anti-Aliasing abgeleitet werden kann.

Sämtliche obengenannten Methoden weisen hochparallele Berechnungslasten auf. Daraus folgend, präsentieren wir im Zuge der gesamten Dissertation deren Abbildung auf massivparallele Hardwarearchitekturen in der Form von Grafikprozessoren. Da unsere Ansätze nicht auf herkömmlichen Grafikpipelines umzusetzen sind, implementierten wir sie mithilfe von generellen Berechnungskonzepten. Dies schlägt sich in reduzierten Laufzeiten nieder und ermöglicht die Interaktivität all unserer Methoden.

Acknowledgements

FIRST of all, I would like to thank my supervisors Prof. Michael Wimmer and Stefan Jeschke. If it were not for them and their willingness to employ a physicist for research in computer graphics, I would not have been able to pursue this degree. I am grateful for their supervision and guidance especially in my first months of orientation and their continued help.

I want to thank my close collaborator Gabriel Mistelbauer for inspiring discussions and for unconditional commitment especially at submission deadlines. Without support of my collaborators Reinhold Preiner, Przemyslaw Musialski, and Michael Guthe, this thesis would not have been possible. My thanks also go to my collaborators during my former and continued research: Johanna Schmidt, Paul Guerrero, María del Carmen Calatrava Moreno, Ralf Habel, Alexey Karimov, Károly Zsolnai, Ralf Habel, Stefan Bruckner, and Eduard Gröller. Working with you is and was a pleasure.

I owe the staff of our institute, especially Andreas Weiner and Stephan Bösch-Plepelits, for excellent technical assistance, and Anita Mayerhofer-Sebera, Sow Wai (Tammy) Ringhofer, and Andrea Fübi for their invaluable help throughout the years. I am also thankful to the head of our institute, Werner Purgathofer, for creating such an enjoyable work environment. I acknowledge Gernot Ziegler from NVidia Austria provided great GPU-related insights.

I want to thank my parents, Hermine and Walter Auzinger, for their support throughout my years of study. Very special thanks to my girlfriend María del Carmen Calatrava Moreno, who lovingly and patiently carried me through the years of my thesis and provided invaluable emotional as well as professional support. Te amo, Mamen.

I was funded during my doctoral studies by the FWF projects *Modern Functional Analysis in Computer Graphics* (FWF P23700-N23) and *Detailed Surfaces for Interactive Rendering* (FWF P20768-N13).

Contents

CHAPTER 1

Introduction

THE generation of images can be regarded as the foundation on which computer graphics is built upon. From physically based light-transport simulation for photo-realistic imagery, over their fast approximations to enable real-time performance, up to the depiction of abstract data with visualizations, an image is nearly always the final output. To match this output with display hardware—which, since decades, uses a regular arrangement of light-emitting picture elements (i.e., pixels) to generate the visual impression—a raster format has to be chosen. Many common image formats (e.g., Joint Photographics Experts Group (JPEG), Portable Network Graphics (PNG), and Graphics Interchange Format (GIF)) and almost all video format are, therefore, regular grids of color values.

There are, however, fundamental mismatches between most of the input data to an image-generation process—also called *rendering* process—and its output. First, the input is often given in non-raster formats such as vector formats. An examples for such format is text, which is commonly represented by non-linear curves that describe the outlines of letters and symbols. Another example is the scene geometry for light-transport simulations, which is represented by the objects' surfaces using meshes of polygons or non-linear patches. Even if given in raster format, such as texture maps, the regular grids of input and output hardly ever coincide. Thus, a format conversion has to be performed in a process that is commonly referred to as *rasterization* in the computer graphics community and as *sampling* in other fields.

This leads to an inevitable loss of information, since the richness of the vector information cannot be represented by the discrete samples, which are located at the grid points of the raster output. As described in seminal works in the theory of signal processing, there are fundamental limitations associated with sampling processes, which can manifest in severe data distortion. Subsumed under the term *aliasing*, a class of these deficiencies arise from an inherent limit on the image sharpness that a given regular grid can still represent. To reduce aliasing artifacts to a minimum, image details beyond this sharpness limit have to

be removed in a process that is commonly called *anti-aliasing*. The first focus of this thesis is concerned with the task of how anti-aliasing can be performed such that the output quality of the rasterization process in maximized. As we will show, this requires a special class of filters and symbolic mathematical computations for their application.

The second mismatch between input and output data in image-generation tasks is associated with their respective dimensions. This is evident from the fact that three-dimensional scene input has to be projected onto a two-dimensional output image. Even the conversion of two-dimensional vector graphics to two-dimensional raster images exhibits this complication, since vector graphics are usually organized in layers, where the content of a given layer is occluded by the content of other layers above it. The technique on how to resolve the *visibility* of those shapes or scene elements is intimately linked with the chosen anti-aliasing approach. If the consequence of aliasing is ignored or if a dense intermediate raster grid is used to increase the output quality, the scene visibility can be determined locally for each sample location in the raster grid. Highest-quality anti-aliasing, in contrast, requires an exact vector-format representation of the scene visibility as a basis for the subsequent rasterization process. For applications with different requirements in terms of execution time and output quality, we investigate both approaches in a second focus of this thesis.

All aforementioned computation tasks—as well as image generation in general—are inherently parallel, in the sense that the color of each pixel of the raster output is generated in a similar fashion and largely independently from each other. Apart from algorithmic considerations, in the sense that acceleration data structures and cached results can be efficiently shared among different samples, this large amount of parallelism enables the efficient use of parallel hardware architectures. In fact, it was the widespread use of rasterization in computer games and other visually demanding real-time applications that spawned a whole industry of parallel co-processors. Nowadays, they can be found in nearly all consumer computing devices in the form of graphics chips. We map our algorithms to such parallel computing models, which allows us to investigate their performance characteristics with actual implementations. This constitutes the third focus of this work.

1.1 Motivation

This thesis is motivated by challenges both in applied and fundamental computer graphics, which we review in this section. Consult Table 1.1 for an accompanying overview.

For a medically relevant use case concerned with the radiological examinations of blood-vessel interiors, we employ sampling-based anti-aliasing methods to minimize computational complexity and to guarantee interactive performance. In this work, we resolve the complex visibility of mutually occluding vessels and provide a smooth cut into the surrounding tissues. This enables an efficient examination of potential pathologies by domain experts, since we show as many blood vessels as possible for each view direction, while still leaving sufficient cues to judge the depth ordering of the vessels. Technically,

Technique	Anti-Aliasing Strategy	
	Visibility	Shading
Curved Surface Reformation (Chapter 3)	Sampled	Supersampled
Prefiltering on Polytopes (Chapter 4)		Prefiltered
Exact Parallel Visibility (Chapter 5)	Prefiltered	Prefiltered
Non-Sampled Anti-Aliasing (Chapter 6)	Prefiltered	Sampled

Table 1.1: Overview of the various anti-aliasing strategies that were used throughout the work presented in this thesis. In the applied setting of medical visualization, where interactive feedback is paramount, sample-based methodologies were employed due to their lower computation cost. Our fundamental research into ground-truth anti-aliasing leads to the use of prefiltering first for shading and then for visibility and shading. For various shading models, prefiltering is not possible due to mathematical reasons and sampling is provided as an replacement.

this is achieved by extruding surfaces away from the vessels' centerlines and by modulating their depth values to minimize the occlusions of vessels further in the back. Furthermore, we provide a Level of Detail (LOD)-based smoothing of the extruded surfaces to counteract their possible distortions in the surrounding tissues. In this approach, we employ sampling to resolve the visibility and supersampling to filter the imaging data both in- and outside of the blood vessels. While sampling-based visibility methods generally suffer from poor anti-aliasing quality—especially at object silhouettes—this does not occur in this setting, as silhouttes are overdrawn for application-specific reasons. Thus, we chose the conceptually simpler approach that, at the same time, requires less computational effort and enables the vital interactive response of our visualization framework.

The remainder of the thesis is concerned with the opposite scenario of obtaining the highest possible anti-aliasing quality. By developing a general theory for mathematically exact filtering, we are able provide a ground-truth solution for anti-aliased rasterization. This approach enables the comparison of various approximation to anti-aliasing—of which there exists a rich body of work—against an objective reference. Furthermore, it allows the comparison of different filter functions on scenes with high anti-aliasing requirements. With sampling-based approximations, the effects of different filters is often hidden by the overwhelming aliasing noise. From a technical point of view, this result is achieved by a combination of symbolic integration—to evaluate the convolution integrals of the shading function with the filter kernel—and computational geometry—to exactly remove those parts of the scene that are occluded by objects in front of them. For shading functions

that do not permit symbolic integration, we additionally present a scheme on how to combine exact visibility filtering with sampled shading.

The implementation of the aforementioned works on general-purpose graphics hardware has several motives. Since all our algorithms are highly parallelizable, it is possible to efficiently utilize the computational resources of such hardware architectures. This leads to an improved performance in terms of runtime and energy efficiency. Adhering to the parallel computing model of graphics processor enforces parallelization strategies that are also valid for different massively parallel architectures, such as many-core coprocessors. Thus, our strategies for data reuse, reducing the branching of code paths, and the efficient use of parallel primitives are also valid for a wide range of different applications.

1.2 Contributions

The following contributions are made in this thesis:

- A parallel algorithm to resolve the exact visibility of surfaces that are extruded outwards from central 3D curves and orthogonal to an arbitrary view direction. Conceptually similar to the rasterization-based construction of Generalized Voronoi Diagrams (GVDs) (Hoff et al. 1999), we use a non-standard depth function to minimize mutual occlusion. Employed in medical visualization, this approach allows, for the first time, the depiction of multiple overlapping blood vessels from arbitrary view directions with automatically generated cut-aways.

  This contribution was published as

  > T. Auzinger, G. Mistelbauer, I. Baclija, R. Schernthaner, A. Kochl, M. Wimmer, M. E. Groller, and S. Bruckner (2013). „Vessel Visualization using Curved Surface Reformation“. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12, pp. 2858–2867. ISSN: 1077-2626. DOI: 10.1109/tvcg.2013.215

  in a joint first authorship with Gabriel Mistelbauer, where I contributed the mathematical theory, while Gabriel Mistelbauer created the implementation and conducted the evaluation, where he was aided by the physicians. The last three authors provided editorial support. An exposition can be found in Chapter 3.

- A parallel method to resolve the exact visibility of triangular meshes. We present a novel edge-triangle intersection routine to perform hidden-line elimination. Hidden-surface elimination is then achieved with our new boundary completion algorithm. The polygonal visible regions of each triangle are reported as a set of line segments that constitute the region's boundary. This vector-format data can be directly used to define domains of integrations.

  The publication

> T. Auzinger, M. Wimmer, and S. Jeschke (2013). „Analytic Visibility on the GPU“. in: *Computer Graphics Forum* 32.2pt4, pp. 409–418. ISSN: 0167-7055. DOI: 10.1111/cgf.12061

documents this contribution. Apart from editorial support by Michael Wimmer and Stefan Jeschke, and result scenes by Stefan Jeschke, I created this work. An in-depth description is presented in Chapter 5.

- An analytic method for prefiltered sampling of linear functions defined on polytopes. A wide range of radially symmetric filters is supported by closely approximating them with a polynomial of arbitrary order. Apart from a closed-form formulation in n dimensions, we specifically allow for high-quality rasterization of 2D polygons and 3D polyhedra to regular and non-regular grids. Together with the visibility routine above, fully prefiltered 3D to 2D rasterization is supported.

  We published this contribution as

  > T. Auzinger, M. Guthe, and S. Jeschke (2012). „Analytic Anti-Aliasing of Linear Functions on Polytopes“. In: *Computer Graphics Forum* 31.2pt1, pp. 335–344. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03012.x

  where Michael Guthe contributed the major part of the filtering theory while Stefan Jeschke provided the DirectX implementation and wrote parts of the articles. The CUDA implementation, the closed-form solutions, the evaluation, and the main part of the article were done by me. It builds the basis of Chapter 4.

- A technique to conventionally sample the values of an arbitrary function on polytopes while still providing analytic prefiltering of the polytope's characteristic function. In the context of rasterization, this allows for near-perfect edge anti-aliasing in combination with conventional shaders.

  Published as

  > T. Auzinger, P. Musialski, R. Preiner, and M. Wimmer (2013). „Non-Sampled Anti-Aliasing“. In: *Vision, Modeling & Visualization*. Ed. by M. Bronstein, J. Favre, and K. Hormann. VMV '13. The Eurographics Association, pp. 169–176. DOI: 10.2312/PE.VMV.VMV13.169-176

  I was assisted in writing the article by Reinhold Preiner and Przemyslaw Musialski. Michael Wimmer provided discussions and editorial support. The remainder was created by myself. Chapter 6 presents this contribution.

- A mapping of all aforementioned techniques to a massively parallel hardware architecture. We provide parallelization strategies for efficient execution on a large number of Single Instruction Multiple Data (SIMD) processing units, typical for Graphics Processing Units (GPUs) and many-core coprocessors. Furthermore, we present implementations on graphics hardware using a general-purpose computing

Application Programming Interface (API) together with performance evaluations on recent graphics cards.

Chapters 3-6 cover different aspects of this mapping, with the most in-depth discussion presented in Chapter 5.

1.3 Organization

The thesis is organized into the following chapters:

Chapter 2 introduces fundamental concepts and provides the context of this thesis by embedding it into related works.

Chapter 3 presents a reformation framework for radiological inspections of blood-vessel interiors. By resolving the complex visibility of multiple local surfaces around the vessel centerlines and by creating automatic cutaways, mutual vessel occlusions are reduced to a minimum and most of the vasculature is shown at once. The use of sampling for visibility resolution is presented together with supersampled shading and post-processing anti-aliasing. Based on an evaluation by several domain experts, the usefulness of our approach is demonstrated.

Chapter 4 initiates the part on prefiltered anti-aliasing. Closed-form solutions to convolution integrals of polytopes with radial filter kernels are presented in arbitrary dimensions. This is achieved by appropriate subdivisions of the integration domain, which is the intersection of the polytope with the hyperspherical filter support. An implementation of 2D and 3D rasterization on graphics hardware is presented and its performance characteristics evaluated.

Chapter 5 augments the shading prefiltering of the previous chapter with exact visibility resolution in the case of 3D to 2D rasterization. By extending previous results from computational geometry, an algorithm for hidden-surface elimination is presented, which is tailored to massively parallel SIMD architectures, such as GPUs. Apart from a detailed description of the various stages of the method—edge intersection, hidden-line elimination, and boundary completion—an implementation on graphics hardware is introduced and benchmarked.

Chapter 6 builds on the previous chapters and merges visibility prefiltering with sampled shading. The result is a novel rasterization pipeline that combines ground-truth edge anti-aliasing with arbitrary shading functions and effects. It shows how to replace conventional depth buffering with exact hidden-surface elimination while still leaving vertex and fragment shader largely unaltered. An implementation on GPUs using DirectX and Compute Unified Device Architecture (CUDA) is presented and evaluated.

Chapter 7 summarizes the thesis with a discussion section and outlines possible future extensions and applications of the presented methods.

1.4 Publications

This thesis is based on the following accepted peer-reviewed publications:

Auzinger, T., G. Mistelbauer, I. Baclija, R. Schernthaner, A. Kochl, M. Wimmer, M. E. Groller, and S. Bruckner (2013). „Vessel Visualization using Curved Surface Reformation“. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12, pp. 2858–2867. ISSN: 1077-2626. DOI: 10.1109/tvcg.2013.215.

Auzinger, T., P. Musialski, R. Preiner, and M. Wimmer (2013). „Non-Sampled Anti-Aliasing“. In: *Vision, Modeling & Visualization*. Ed. by M. Bronstein, J. Favre, and K. Hormann. VMV '13. The Eurographics Association, pp. 169–176. DOI: 10.2312/PE.VMV.VMV13.169-176.

Auzinger, T., M. Wimmer, and S. Jeschke (2013). „Analytic Visibility on the GPU“. In: *Computer Graphics Forum* 32.2pt4, pp. 409–418. ISSN: 0167-7055. DOI: 10.1111/cgf.12061.

Auzinger, T., M. Guthe, and S. Jeschke (2012). „Analytic Anti-Aliasing of Linear Functions on Polytopes“. In: *Computer Graphics Forum* 31.2pt1, pp. 335–344. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03012.x.

During the time period of this thesis, the following unrelated peer-reviewed articles were published:

Ilcik, M., P. Musialski, T. Auzinger, and M. Wimmer (2015). „Layer-Based Procedural Design of Facades“. In: *Computer Graphics Forum* 34.

Jimenez, J., K. Zsolnai, A. Jarabo, C. Freude, T. Auzinger, X.-C. Wu, J. von der Pahlen, M. Wimmer, and D. Gutierrez (2015). „Separable Subsurface Scattering“. In: *Computer Graphics Forum*. ISSN: 0167-7055. DOI: 10.1111/cgf.12529.

Guerrero, P., T. Auzinger, M. Wimmer, and S. Jeschke (2014). „Partial Shape Matching Using Transformation Parameter Similarity“. In: *Computer Graphics Forum*. ISSN: 0167-7055. DOI: 10.1111/cgf.12509.

Schmidt, J., R. Preiner, T. Auzinger, M. Wimmer, M. E. Gröller, and S. Bruckner (2014). „YMCA - Your Mesh Comparison Application“. In: *IEEE Conference on Visual Analytics Science and Technology*. IEEE VAST '14. IEEE.

Furthermore, the following additional publications were produced in the period of this thesis:

Auzinger, T. and M. Wimmer (2013). „Sampled and Analytic Rasterization“. In: *Vision, Modeling & Visualization Posters*. Ed. by M. Bronstein, J. Favre, and K. Hormann. VMV '13. The Eurographics Association, pp. 223–224. DOI: 10.2312/PE.VMV.VMV13.223-224.

Calatrava Moreno, M. d. C. and T. Auzinger (2013). „General-Purpose Graphics Processing Units in Service-Oriented Architectures“. In: *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*. SOCA '13. IEEE, pp. 260–267. isbn: 978-1-4799-2701-2. doi: 10.1109/soca.2013.15.

Auzinger, T., R. Habel, A. Musilek, D. Hainz, and M. Wimmer (2012). „Geiger-Cam: Measuring Radioactivity with Webcams“. In: *ACM SIGGRAPH 2012 Posters*. SIGGRAPH '12. ACM Press, 40:1–40:1. isbn: 978-1-4503-1682-8. doi: 10.1145/2342896.2342949.

CHAPTER 2

Related Work and Foundations

In this chapter, we provide the conceptual and mathematical foundation of sampling, filtering, and anti-aliasing in Section 2.1. In Section 2.2, an overview of parallel computing architectures with a focus on graphics hardware is presented. Furthermore, we provide the context to this thesis with the related work sections on anti-aliasing of shading and visibility signals (see Section 2.1.5) and on the use of general-purpose computation on Graphics Processing Units (GPUs) in the field of computer graphics (see Section 2.2.1).

2.1 Sampling and Filtering

Most output modalities in visual computing expect a discrete data format. Displays show content as a regular grid of color pixels while 3D printers recreate sliced or voxelized representations of the actual model. The format of input data, in contrast, is often (piecewise) continuous. Most 2D and 3D models are represented as piecewise linear or non-linear patches, such as polygonal or polyhedral meshes, splines, or explicit and implicit parametrizations. Scalar and vector functions to describe volumetric and surface effects, such as illumination models, are often given as analytic expression. And even if input is given in a discrete format, it will rarely align perfectly with the output discretization.

Being a fundamental operation in many applications—far beyond computer graphics—, the conversion process from a continuous to a discrete representation is subsumed under the term *sampling* and has its theoretical foundations in the field signal processing. For already discrete data, *resampling* is possible by creating an intermediate continuous representation. In the following, the fundamentals of one- and multidimensional sampling and filtering are provided.

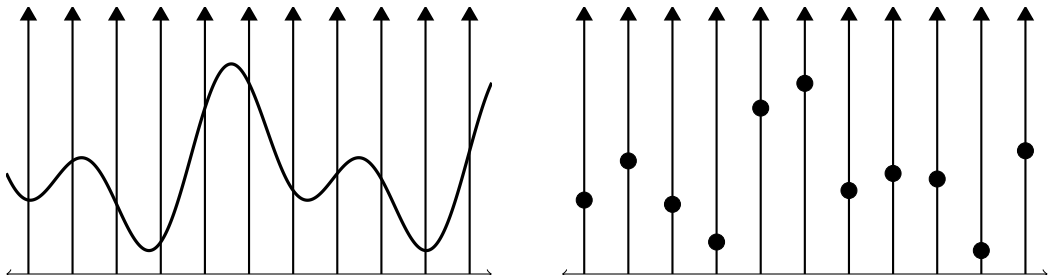


Figure 2.1: Application of the Dirac comb to obtain a regular sampling of a continuous function (left). The function is only evaluated at the location of the delta distributions (right).

2.1.1 One-dimensional Sampling and Filtering

Formally, sampling of a (piecewise) continuous 1D function $f(t)$ at a discrete location t_0 produces the *sample* $f(t_0)$. Using the Dirac delta distribution $\delta(t)$, heuristically given by

$$\delta(t) = \begin{cases} \infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \qquad \text{and} \qquad \int_{-\infty}^{\infty} \delta(t)\, dt = 1,$$

the sampling process can be written in integral form as

$$f(t_0) = \int_{-\infty}^{\infty} f(t)\delta(t - t_0)\, dt, \tag{2.1}$$

where the *point sample* $f(t_0)$ and the *impulse sample* $f(t)\delta(t-t_0)$ can be used interchangeably. In the following, we will use the impulse sample notation as it is more suitable to the further mathematical exposition. Equation 2.1 has the form of a convolution and we will use the following shorthand:

$$(f \star \delta)(t_0) = \int_{-\infty}^{\infty} f(t) \underbrace{\delta(t_0 - t)}_{=\delta(t-t_0)}\, dt. \tag{2.2}$$

Note that the definition and correct usage of distributions such as δ requires a considerable mathematical framework and we refer to the original works of Sobolev (1936) and Schwartz (1951–1957) as well as to a modern textbook (Strichartz 2003) for an overview.

A common use case in computer graphics is the sampling at regular intervals where the spacing T between adjacent samples is constant. As illustrated in Figure 2.1, the sampled signal f_T is thus given as

$$f_T(t) = \sum_{k=-\infty}^{\infty} f(t)\, \delta(t - kT)$$

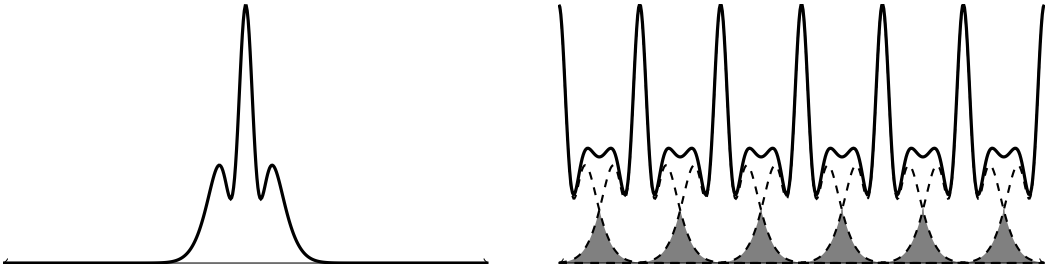


Figure 2.2: The aliasing effect in frequency space. The spectrum of a continuous signal is shown on the left. The spectrum of a regular sampling of this signal is shown on the right. It consists of translated copies—the aliases—of the signal's original spectrum (dashed), which overlap (gray area) if the sample spacing is too small. This causes a distortion of the samples signal's low frequencies by aliases at higher frequencies.

and we denote the sampling function as

$$\text{Ш}_T(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT). \tag{2.3}$$

In the literature, Ш has a multitude of names, e.g., impulse train, Shah function, bed of nails, replicating symbol, or Dirac comb and in this work we refer to it as *comb*. A fundamental principal of sampling becomes evident when transforming our setting into frequency space. By applying the Fourier transform and its inverse defined as

$$\begin{aligned}\widehat{f}(\xi) &= \int_{\infty}^{\infty} f(t)\, e^{-2\pi i t\xi}\, dt \\ \check{f}(t) &= \int_{\infty}^{\infty} f(\xi)\, e^{2\pi i \xi t}\, d\xi\end{aligned}$$

and using the identity $\widehat{\text{Ш}_T} = \frac{1}{T}\text{Ш}_{-\frac{1}{T}}$, we obtain the frequency representation of regular sampling as

$$\begin{aligned}\widehat{f_T}(\xi) = \widehat{f\,\text{Ш}_T}(\xi) &= \frac{1}{T}\left(\widehat{f} \star \widehat{\text{Ш}_{-\frac{1}{T}}}\right)(\xi) \\ &= \frac{1}{T}\sum_{k=-\infty}^{\infty} \widehat{f}\left(\xi + \frac{k}{T}\right),\end{aligned} \tag{2.4}$$

where we used the convolution theorem stating that $\widehat{f \star g}(\xi) = \widehat{f}(\xi)\,\widehat{g}(\xi)$. This computation shows that regular sampling in frequency space amounts to the infinite sum of translated copies, called *aliases*, of the spectrum $\widehat{f}$ of our input function f.

If the support of the transformed signal $\widehat{f}$ has a width exceeding $\frac{1}{2T}$, the aliases overlap as shown in Figure 2.2. This causes a distortion of the lower frequencies by the higher frequencies of the aliases, i.e., the frequency content of the original signal above $\frac{1}{2T}$ is

introduced as low-frequency patters in the sampled data. This process is commonly referred to as *aliasing* and one focus of this thesis is concerned with methodologies to combat it, i.e., to perform *anti-aliasing*.

The Nyquist-Shannon sampling theorem (Nyquist 2002; Shannon 1998) is a direct consequence from this fact and states that if a function contains no frequencies higher than $\frac{1}{2T}$, it is completely determined by a regular sampling with period T. Such functions are referred to as *band-limited* and they can be perfectly reconstructed from the sampling by a convolution with the sinc function (Whittaker 1915), i.e,

$$f(t) = \left(f_T \star \operatorname{sinc}\left(\tfrac{\cdot}{T}\right)\right)(t).$$

Note that the sampling theorem is a sufficient condition but not a necessary one if further restriction apply to the original function. The field of compressed sensing (Candes et al. 2006) provides methods for sampling with a lower rate at the cost of a more complex reconstruction. While these approaches have their applications in image generation (P. Sen and Darabi 2010), we do not elaborate on them in the scope of this thesis.

In many application—computer graphics among them—the input signal cannot be assumed to be band-limited. Text, for example, where letters are defined by their continuous outline, the signal exhibits a discontinuity when crossing the boundary. Formally, this can be illustrated by transforming the discontinuous rectangle function $\Pi_w(t)$ of width w, given by

$$\Pi_w(t) = \Pi_1\left(\frac{t}{w}\right) = \begin{cases} 0, & |t| > \frac{w}{2} \\ \frac{1}{2w}, & |t| = \frac{w}{2} \\ \frac{1}{w}, & |t| < \frac{w}{2}, \end{cases} \tag{2.5}$$

into frequency space to obtain

$$\widehat{\Pi_w}(\xi) = w \operatorname{sinc}(w\xi) = \frac{\sin(w\pi\xi)}{\pi\xi}.$$

Since $\operatorname{sinc}(t)$ has unbounded support (i.e., $\nexists R > 0, \forall |t| > R : \operatorname{sinc}(t) = 0$), arbitrarily large frequencies contribute to the spectrum of $\Pi(t)_w$.

As a consequence, it is not possible to sample general signals without experiencing aliasing. By sacrificing the high frequency content of the input signal, it is however possible to perform anti-aliasing. This is achieved by removing frequencies above the permissible band-limit from the input signal with the help of a *low-pass filter* in frequency space. The ideal low-pass filter is the rectangle function—also called box-filter—of width $\frac{1}{T}$ as shown in Figure 2.3. By multiplication of the Fourier transformed input signal with the box-filter, all frequencies above the limit of $\frac{1}{2T}$ are removed and aliasing is eliminated. Applying the inverse Fourier transform gives the action of the filtering process on the actual signal as

$$\widehat{\widehat{f}\,\Pi_{\frac{1}{T}}} = f \star \frac{1}{T} \operatorname{sinc}\left(\frac{\cdot}{T}\right). \tag{2.6}$$

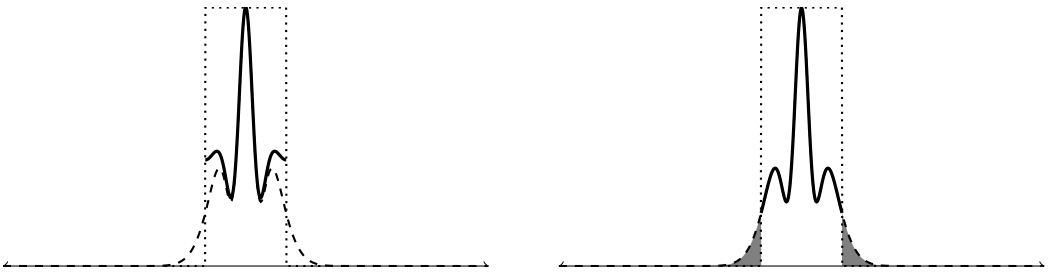


Figure 2.3: Effects of low pass filtering. (Left) If a signal exceeds the band limit imposed by the sampling period, overlaps with aliases occur and the spectrum of the sampled signal (solid) does not coincide with the spectrum of the signal (dashed). (Right) A low pass filter (dotted) removes the offending frequencies (gray) and eliminates aliasing.

As stated by the convolution theorem, the product of the transformed signal with a low-pass filter in frequency space translates to a convolution filtering of the actual signal with the inversely transformed filter. Thus, mathematically perfect anti-aliasing of a regularly sampled signal can be achieved by a convolution with a sinc filter of appropriate scale. Further details on this observation can be found in the textbook of Bracewell (2000).

From a computational viewpoint, however, this result has several significant shortcomings due to the nature of the sinc function. As already mentioned, its support is unbounded. This makes it computationally impossible to actually evaluate the convolution in Equation 2.6 as the domain of the integration is infinitely large. As a further complication, the integration over the function is only finite, if the positive and negative lobes are allowed to cancel each other out. The integral over only the positive (resp. negative) parts diverges to positive (resp. negative) infinity. It is possible to create signals that exhibit an arbitrarily large positive and negative response by this filter at certain sampling locations, which is highly unsuitable for many applications in computer graphics (e.g., displaying colors of bounded range).

A multitude of approximations to the sinc filter were proposed that try to find a trade-off between computationally convenience and anti-aliasing performance. The simplest approaches approximate the main lobe of the sinc function with a similar function of finite support. Often-used examples are the aforementioned box filter as well as tent and Gaussian filters. A different approach in signal processing is to design *windowing functions* with finite support and apply them to the sinc filter. A famous example is the Lanczos window (Duchon 1979) defined as

$$w_{\text{Lanczos}} = \operatorname{sinc}\left(\frac{x}{a}\right)$$

where a determines the width of the window. The actual filter is then given as a multiplication of the sinc filter with this window, i.e., $\frac{1}{T}\operatorname{sinc}\left(\frac{t}{T}\right)\operatorname{sinc}\left(\frac{t}{aT}\right)$. Popular choices are $a = 2, 3$ which include—apart from the main lobe—the first negative lobes

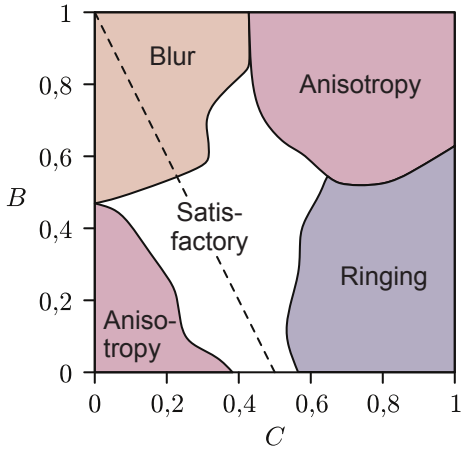


Figure 2.4: Perceptional properties of the Mitchell and Netravali filter family. Only a subset of parameter pairs B and C yielded satisfactory filter functions, as determined by a user study. Extremal cases on the diagonal suffer from exceeding anisotropy, i.e., differences in how orthogonal directions in the image are filtered. On the counterdiagonal, blur manifests in oversmoothing of the image while ringing introduces unwanted oscillations near edges or regions with large gradient. Taken from (Mitchell and Netravali 1988).

or the first negative and positive lobes. Note that for increasing parameter a both the computational complexity as well as the anti-aliasing quality increases. A third category are perceptional motivated filters, such as the filter family designed by Mitchell and Netravali (1988). For a two-parameter family of cubic splines, they determined which parameter pairs yield the perceptionally best filter performance as illustrated in Figure 2.4.

All these approximation suffer from a range of artifacts with *overblurring* and *ringing* being the most perceivable. In general, purely positive filters, such as the Gaussian or box filters, tend to suppress frequencies below the band limit too much and introduce excessive blurring of the signal. Filters with negative lobes, such as the Lanczos filters, tend to overemphasize gradients and cause oscillation artifacts near edges. While a rich theory on these trade-offs exists in the field of signal processing, filter choice in computer graphic is usually subject to personal preferences (Blinn 1989).

2.1.2 Multi-dimensional Sampling and Filtering

So far, we discussed the filtering of one-dimensional signals. However, most applications in computer graphics exhibit a higher dimensionality, such as common rasterization or convolution-based image filtering (2D), voxelization and motion-blur (3D), defocus blur (4D) and bilateral filtering (5D). Some filtering operations are even formulated in dozens of dimensions with Non-Local-Means filtering (Buades et al. 2005) being a prominent example. Potentially, a multi-dimensional signal can be sampled with

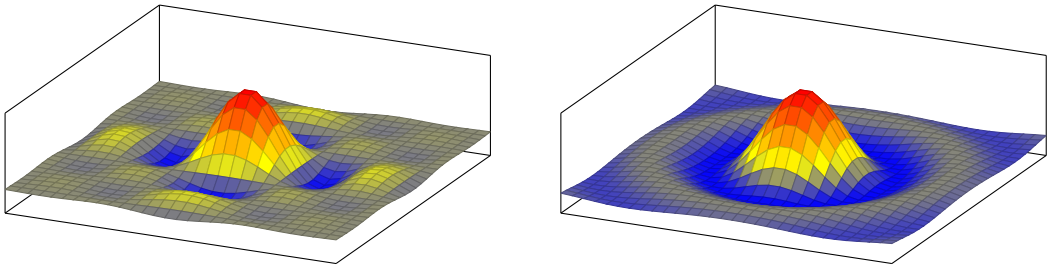


Figure 2.5: Ideal low pass filters in 2D. For a separable filter model, the ideal low pass filter in the spatial domain is the tensor product of two sinc functions (left). The ideal radial low pass filter is given in terms of Gamma and Bessel functions (right).

different methodologies in each dimension, which can be frequently observed when conceptual differences exist between the dimensions (e.g., time and space). In this thesis, we are predominately concerned with n-dimensional spatial sampling and we treat all dimensions identically. Furthermore, we will use regular Cartesian grids as sampling patterns throughout this thesis but we note that most concepts of these works can be applied to arbitrary sample locations.

When extending the formalism of 1D filtering to multiple dimensions, two generalizations are obvious choices: tensor products and radial extensions. The former takes a 1D filter function $g(t)$ and builds a separable n-dimensional filter function $g_{\boxtimes}(\boldsymbol{t})$ by

$$g_{\boxtimes}(\boldsymbol{t}) = g_{\boxtimes}(t_1, t_2, \ldots, t_n) = g(t_1)\, g(t_2) \cdots g(t_n)$$

whereas the latter generates a radial n-dimensional filter function $g_{\odot}(\boldsymbol{t})$ by

$$g_{\odot}(\boldsymbol{t}) = g_{\odot}(t_1, t_2, \ldots, t_n) = g_{\odot}(\|\boldsymbol{t}\|) = g\left(\sqrt{t_1^2 + t_2^2 + \cdots + t_n^2}\right).$$

We get the ideal low pass filter in this setting by using the box filter $\Pi_w(\xi)$ (2.5) as 1D filter function in frequency space. For both generalization to n dimension, the corresponding convolution filter in the spatial domain can be obtained by the inverse n-dimensional Fourier transform. In the separable case, the convolution filter is given as the tensor product of one-dimensional sinc filters, i.e.,

$$\widetilde{\Pi_{\boxtimes, \frac{1}{T}}}(\boldsymbol{t}) = T^n \operatorname{sinc}\left(\frac{t_1}{T}\right) \operatorname{sinc}\left(\frac{t_2}{T}\right) \cdots \operatorname{sinc}\left(\frac{t_n}{T}\right)$$

while the radial extensions produces

$$\widetilde{\Pi_{\odot, \frac{1}{T}}}(\boldsymbol{t}) = J_{\frac{n}{2}}\left(\frac{\pi\|\boldsymbol{t}\|}{T}\right) (2T\|\boldsymbol{t}\|)^{-\frac{n}{2}} \tag{2.7}$$

with $\Gamma(t)$ denoting the Gamma function and $J_\nu(t)$ the Bessel function of the first kind. We present a derivation in Appendix B. Further details can be found in the work of James (1995) and both filters are illustrated in Figure 2.5.

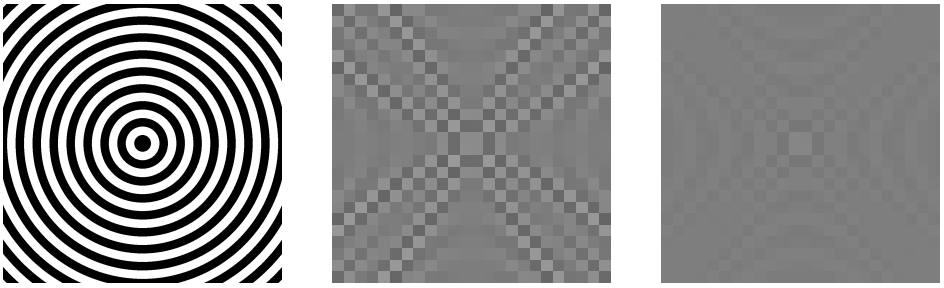


Figure 2.6: Comparison of radial and separable 2D filters. A circular pattern (left) with a frequency slightly above the Nyquist limit of the output raster grids. A separable Lanczos-windowed sinc filter (center) exhibits strong anisotropy along the diagonal directions. With the equivalent radially symmetric filter (right) these artifacts are greatly reduced.

On a regular Cartesian grid the tensor product of sinc filters is the *mathematically ideal* n-dimensional filter as its support is the full Brillouin zone of the reciprocal lattice (Petersen and Middleton 1962). The computational complexity of evaluating such a filter is also considerably lower, since an n-dimensional convolution can be decomposed in n one-dimensional convolutions due to the separability of the filter. It shows strong anisotropy, however, since the sample density is higher along the diagonals, and less blurring happens along these directions. From a perceptional standpoint, this is highly undesirable as one would expect a filtered radially symmetric signal to show the same characteristics in all directions (see Figure 2.6). The user studies of Mitchell and Netravali (1988) support this assumption, since only filters with low anisotropy were deemed satisfactory (see Figure 2.4). Thus, we will predominately employ radial filters in this thesis.

2.1.3 Convolution Computation

The approach on how to evaluate a filter convolution is a key differentiator of many works on anti-aliasing in computer graphics. For a sample at location $\boldsymbol{t}_0$ and a filter function $g(\boldsymbol{t})$ we obtain its sampled value by the filter convolution

$$f(\boldsymbol{t}_0) = \int_{\mathbb{R}^n} f(\boldsymbol{t})\, g(\boldsymbol{t}_0 - \boldsymbol{t})\, d\boldsymbol{t}. \tag{2.8}$$

For most computational purposes, this convolution has to be evaluated numerically and as illustrated in Figure 2.7, four main categories can be identified:

Sampling The simplest method is to ignore the filter contribution and to *directly sample* the signal value at $\boldsymbol{t}_0$. As already mentioned before, this approach suffers from the aliasing artifacts that we tried to combat with the filtering process. Common rasterization and ray tracing are based on this principle, where each pixel of the output image is assigned the color obtained at its center.

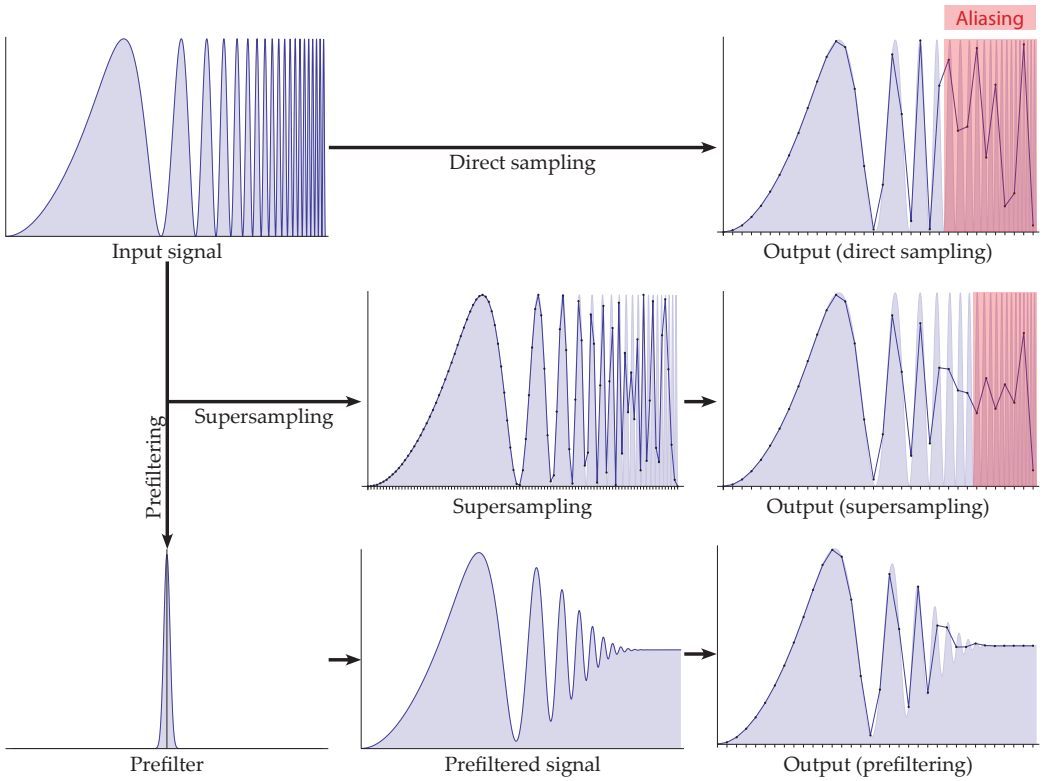


Figure 2.7: The main categories of filter convolution computations. By *directly sampling* an input signal, severe aliasing artifacts can be observed, if the signal frequencies exceed the band limit as imposed by the sample spacing. *Supersampling* utilizes a dense intermediate sampling and applies resampling to reduce the sampling resolution to the desired value. Although to a lesser extent, this approach still suffers from aliasing due to the band limit of the intermediate sampling. A perfect result can be achieved by continuously convolving the input signal with a suitable *prefilter*. This completely eliminates frequencies beyond the band limit and subsequent sampling is free of aliasing artifacts. A fourth category in the form of *reconstruction* post-processing takes the output of either sampling- or supersampling-based methods and applies heuristic transformations to recover parts of the signals that are lost due to aliasing.

Supersampling A better approach is to approximate the filter convolution by a weighted averaging of multiple samples inside the filter footprint, a process generally called *supersampling* or *postfiltering*. Many well-known anti-aliasing methods in rasterization are based on this principle, such as Supersample Anti-Aliasing (SSAA). While conceptually simple and fast to evaluate, supersampling still suffers from aliasing; however, the threshold beyond which aliasing occurs is shifted to higher frequencies when compared to naive sampling.

Prefiltering The mathematically exact approach is to compute the closed-form solution of (2.8) and to sample the filtered signal at the desired locations. This approach removes all aliasing to the extent that the chosen filter function is capable of. As a downside, the evaluation of the potentially lengthy closed-form solutions—assuming that they can be obtained in the first place—is generally computationally expensive.

Reconstruction Not to be confused with the same concept in general signal processing, where reconstruction denotes the recovery of a continuous signal from its discrete sampling, reconstruction filters in rendering enhance the discrete output of a (super)sampling process by applying heuristic operations. This approach works well in spatial anti-aliasing, since staircase artifacts at object silhouettes can be reduced by edge detection and selective resampling. While this approach is far from a general anti-aliasing algorithm, it is computationally less demanding than supersampling or prefiltering, which makes it a popular choice for real-time applications.

2.1.4 Dimensions in Rasterization

So far, the sampling of arbitrary signals was discussed. Common signals in rasterization exhibit far less degrees of freedom and specialized techniques can take further assumptions on the signal into account to provide a better quality at less computational effort. A first overview of typical signals in rasterization can be obtained by examining the dimensionality of the signal. The most important dimensions are the two spatial image dimensions, the depth dimension of three-dimensional scenes, a temporal dimension, and two lens dimensions to simulate a simplified model of an optical system of the camera. Each dimension has a graphical effect associated with it, which are the image formation itself, the scene visibility, motion blur, and Depth of Field (DOF).

In this thesis, we take the spatial and depth dimension into account, for which we provide both sample-based and prefiltering approaches (see Table 1.1). Motion and defocus blur—which are caused by the temporal and lens dimensions—either do not apply to the application setting or would not allow exact prefiltering due to the non-linearities in the associated integrands. For completeness, we also want to state that this thesis in concerned with rasterization techniques and we do not cover ray- or path-based rendering methodologies (Křivánek et al. 2014) that are primarily used for light-transport simulations to provide global illumination. Thus, we leave aside the additional dimensionalities that arise with secondary or higher light bounces.

2.1.5 Related Work in Rasterization

The general problem of aliasing was introduced to computer graphics by the work of Crow (1977), who gave an enumeration of possible artifacts caused by visibility and shading undersampling. Visibility undersampling can cause staircase artifacts along silhouettes or the disappearance of small objects, whereas shading undersampling can result in low-frequency patterns on distant textures or missing highlights. Similar considerations for general image processing can be found in the works of Schreiber and Troxel (1985). In the following, we provide a historical overview and current related work for the different approaches to perform anti-aliasing. A recent survey was also given by Jiang et al. (2014).

Prefiltering

Shading. A considerable amount of work has been published on 2D analytic anti-aliasing. Early methods by Catmull started of with box filtering (Catmull 1978) and were extended to spherically symmetric filters using look-up tables (Catmull 1984). The latter work builds on a filtering method by Feibush et al. (1980) which uses domain decomposition to evaluate the integrals. Pioneering works treat constant color polygons (Catmull 1978; Kajiya and Ullner 1981) and smoothly shaded polygons are mentioned as a possible extension by Catmull (1984). It should be noted that Kajiya and Ullner (1981) also treats the more general problem of finding an optimal filter for the given raster display. Grant (1985) used a 4D formulation of this problem to model spatial and temporal anti-aliasing of polygons that are transformed via translation and scaling. Before the work described in this thesis, the de-facto state of the art still was an analytic 2D filtering method by Duff (1989). It supports general filter models with polynomial approximations and supports linear functions defined on the polytopes. Unfortunately, the separable 2D formulation is not easily generalizable to three dimensions as the integral complexity grows beyond a manageable level. Later, McCool (1995) proposed simplicial decompositions of shapes for a faster filtering compared to Duff at that time, but an extension to three dimensions is not straightforward as well. Furthermore, Guenter and Tumblin (1996) used quadrature prefiltering to further speed up the process, with analytic aliasing in one direction and sampling in the other. More recently, Lin et al. (2005) proposed an analytic evaluation approach for radially symmetric filters that is in spirit similar to this work, but they do not support linear functions nor negative filter lobes, which are needed for most practical applications (Mitchell and Netravali 1988).

Concerning three dimensional rasterization, i.e., *voxelization*, a distinction has to be made between surface voxelizations (Jones 1996; Kaufman 1987) and solid object voxelizations (Wang and Kaufman 1993). In addition, *binary* voxelization techniques (Hasselgren et al. 2005; Huang et al. 1998; Pantaleoni 2011a; Zhang et al. 2007) only apply a binary value to each voxel, effectively ignoring filtering issues. Several solid voxelization papers apply filtering, either based on integral lookup tables (Wang and Kaufman 1993) and/or on the closest distance of each voxel center to the shape boundary (Sramek and Kaufman 1998; 1999) and applying so-called *oriented box filtering*. Note that all these methods do not compute the exact volumetric filter integral. A recent submission that builds upon the

techniques developed in this thesis presents applications of prefiltered 3D voxelization in computational physics (Powell and Abel 2014).

Simultaneously to the work presented in this thesis, Manson and Schaefer (2011; 2013) published results that tackle aspects of the prefiltered rasterization problem. In their first work, they use a Haar wavelet-based representation to rasterize 2D and 3D shapes. In the 2D case, they rasterize shapes with boundaries defined by splines and in the 3D case, polygonal meshes. While allowing for hierachical rasterization in a Level of Detail (LOD) fashion, their approach is restricted to box filtering and binary shading functions. Their second work omits the hierarchical nature of wavelets and presents a patchwise approximation of arbitrary filters to permit the rasterization of linear color gradients of curved shapes utilizing a scanline methodology.

Visibility. Analytic visibility methods were developed early in the history of computer graphics, with the first hidden-line and hidden-surface-elimination algorithms by Appel (1967) and Roberts (1963). Sutherland et al. (1974) presented an excellent survey of related methods and their close relationship to sorting (Sutherland et al. 1973).

Plenty of early algorithms exist for the elimination of hidden lines (Galimberti 1969; Hornung 1982) or general curves (Elber and Cohen 1990). They build the basis for line-based renderings, with the first halo rendering presented by Appel et al. (1979), and recent developments covered in the course by Rusinkiewicz et al. (2008).

Extensions to analytic hidden-surface elimination were conducted by Weiler and Atherton (1977), who clip polygonal regions against each other until trivial depth sorting is obtained, by Franklin Franklin 1980, who used tiling and blocking faces to establish run-time guarantees, and by Catmull (1978; 1984), who sketched a combination of analytic visibility and integration for Central Processing Units (CPUs). An early parallelization is described by Chang and Jain (1981). McKenna (1987) was the first to rigorously show a worst-case optimal sequential algorithm with $O(n^2)$ complexity in the number of polygons. Improvements were made by Mulmuley (1989) and Sharir and Overmars (1992), who aimed for output-sensitive algorithms, whose run-time complexity depends on the actual number of intersections between the polygons. One of the first parallel algorithms was the terrain visibility method by Reif and S. Sen (1988), later improved by N. Gupta and S. Sen (1998). The general setting was treated by Randolph Franklin and Kankanhalli (1990) but with worst-case asymptotics independent of processor count.

Once hardware limitations disappeared, the focus of the community moved to approximate, sampling-based visibility as documented in the following sections. Nowadays, ray- and path-tracing methodologies predominantly rely on object space visibility, e.g., space partitioning hierarchies, while rasterization mainly uses the z-buffer methodology; an overview was given in the course by Durand (2000).

The computational geometry community showed continued interest in analytic visibility, and in recent years Dévai (2011) gave an easier-to-implement optimal sequential algorithm and an optimal parallel algorithm that runs in $\Theta(\log n)$ using $n^2/logn$ processors in the

Concurrent Read Exclusive Write (CREW) Parallel Random-Access Machine (PRAM) model. However, such optimal algorithms use intricate data structures and are highly non-trivial to implement on actual GPU hardware. Nevertheless, our exact visibility method as presented in Chapter 5 draws several inspirations from this work.

Supersampling

With the success of commodity graphics hardware, the research shifted to approximate sampling-based solutions. Integrated into the graphics pipeline, different strategies were developed to balance quality and performance requirement. Multisample Anti-Aliasing (MSAA) (Akeley 1993) performs supersampling of the projected scene visibility. Shading is only executed once per pixel and per (partially) visible primitive and a final blending incorporates the supersampled visibility information. An optimization was introduced with Coverage Sampling Anti-Aliasing (CSAA) (Young 2006). McGuire et al. (2010) proposed the use of massive sampling for motion and defocus blur effects. A technique to supersample both visibility and shading independently of each other was introduced with *decoupled sampling* (Ragan-Kelley et al. 2011) which is aimed at complex effects such as DOF and motion blur. While initially proposed for future hardware architectures, decoupled sampling was mapped to current GPU designs by Liktor and Dachsbacher (2012) and Clarberg and Munkberg (2014). Further hardware designs for multi-rate and adaptive shading were presented by He et al. (2014) and Clarberg et al. (2014). A recent work on vector graphics rasterization employed efficient sample sharing to enable high quality filtering (Ganacim et al. 2014).

Semi-Analytic Methods

Since the rasterization problem is multidimensional, it is possible to employ different filtering methodologies for different dimensions. Gribel et al. (2010) sampled the spatial dimensions but performs visibility prefiltering along the time axis to enable high quality motion blur for flat shaded triangles. This was extended to line samples in the spatial domain while approximating prefiltering with numeric integration (Gribel et al. 2011). Line samples were further used for Depth of Field (DOF) effects by Tzeng et al. (2012). General frameworks for multidimensional non-point sampling were recently presented by Sun et al. (2013) and Ebeida et al. (2014).

Reconstruction

Many of the aforementioned approaches do not harmonize with the nowadays popular deferred shading techniques (Deering et al. 1988), and, in recent years, screen-space post-processing approaches have been explored intensively. For spatial anti-aliasing, edge-aware smoothing of the output image is performed and the work of Reshetov (2009) on Morphological Anti-Aliasing (MLAA) ignited new interest in this field. Recent implementations on graphics hardware were presented by Chajdas et al. (2011), Jimenez et al. (2012), and Lottes (2009). McGuire et al. (2012) provided an extension for

motion blur. For an overview of this rapidly evolving field we refer to a course on postprocessing anti-aliasing (Jimenez et al. 2011) and the yearly courses on real-time rendering (Tatarchuk et al. 2014).

2.2 Massively Parallel Hardware Architectures

Rendering in general—and rasterization in particular—is an inherently parallel problem. For each sample location of the raster output, essentially the same task has to be executed. Furthermore, these tasks are largely independent from each other. This observation lead to the development of special-purpose parallel hardware with the aim of accelerating rendering tasks.

2.2.1 History and Related Work

While the first graphics hardware relied on vector displays (Sutherland 1964) and wire frame drawings, the technological advances in microelectronics lead to fast frame buffers (S. Gupta et al. 1981) and depth-buffered solid rasterization (Akeley 1993). Single Instruction Multiple Data (SIMD) architectures, where the same instruction is executed on different data elements in parallel, were early found to be a good match for computer graphics tasks (Levinthal et al. 1987; Levinthal and Porter 1984).

The growing popularity of computer gaming and the subsequent demand for accelerated 3D rendering in the late 90s led to the rise of a whole industry sector dedicated to consumer graphics hardware. The leading graphics hardware manufacturer, NVidia and Advanced Micro Devices (AMD), have their roots in this time. Standardized Application Programming Interfaces (APIs), such as the Open Graphics Library (OpenGL) and Microsoft's DirectX, provided a uniform interface for software developers to the underlying hardware and gradually more and more parts of the rasterization pipeline were moved to the hardware. With the rise of programmable vertex and fragment shaders (Lindholm et al. 2001; Mark et al. 2003), where arbitrary instruction can be issued to certain stages of the rasterization pipeline, it became possible to use graphics for general purposes beyond graphics. First attempts consisted of a mapping of general computing problems to graphics primitives, either for specialized setting such as linear algebra (Krüger and Westermann 2003), or as a new programming language (Buck et al. 2004).

With the release of dedicated GPU programming language the research activity into General-Purpose Computing on Graphics Processing Units (GPGPU) increased dramatically. Important initial works are the library for scan-based computing primitives for graphics hardware by Sengupta et al. (2007), a method for GPU-based sorting by Satish et al. (2009), and performance optimizations for dense linear algebra by Volkov and Demmel (2008). Since then, GPUs are used for a wide range of applications and we refer to our survey for an overview (Calatrava Moreno and Auzinger 2013).

Eventually, GPGPU also found its uses in graphics. Both rasterization- (Laine and Karras 2011) and ray-based (Parker et al. 2013) rendering systems were implemented

as well as construction algorithms for acceleration structures (Karras and Aila 2013), voxelizations (Pantaleoni 2011b), and volume rendering (Balsa Rodríguez et al. 2014). We contribute to this line of work by providing GPGPU-based implementations for anti-aliased rasterization and rendering of surface arrangements.

To motivate the design choices for our parallel algorithms in this thesis, we give a short introduction to the hardware architecture of modern GPUs. Note that we forgo an elaboration on a simultaneous hardware development in the form of many-core CPU architectures—represented by the Xeon Phi product series of the Intel Corporation—, which have also been examined for their suitability in ray-tracing (Wald et al. 2014) and acceleration structure generation (Wald 2012). Note that many parallelization strategies apply to both platforms, such as SIMD-conscious memory accesses and branching as well as cache blocking (Satish et al. 2012). In this sense, many of our design decisions would apply to general many-core architectures as well.

2.2.2 Graphics Hardware Architectures[1]

The fundamental design principle that differentiates GPUs from CPUs is how the limited amount of chip estate is distributed among the various functional units. CPUs employ sophisticated control units to perform branch prediction and out-of-order executions to provide maximum performance for a small number of threads. Furthermore, large caches enable data reuse and reduce bandwidth to system memory. GPUs, in contrast, rely on large arithmetics units to provide compute resources to a huge amount of threads. Large caches are replaced by high memory bandwidths to the global memory of the device. These choices determine the matching use cases for each architecture: while CPUs provide superior performance for a low number of highly divergent code paths, parallel workloads with either high compute or memory transfer requirements best fit GPUs.

On the coarsest level, GPUs can be classified as a Single Program Multiple Data (SPMD) architecture according to an extended version of Flynn's taxonomy (Darema et al. 1988; Flynn 1972) (see Figure 2.8). Each Processing Element (PE) executes the same program but has its own data and position in the instruction stream. Note that recent hardware also supports the concurrent launch of sufficiently independent programs. A GPU employs several hundred simultaneously active PEs. In each cycle, several schedulers issue instructions to a small subset of these. This allows an efficient utilization of the available computation units, since PEs that execute high-latency operations (e.g., memory accesses) can idle until these operations are completed. In the meantime, PEs that are ready to execute an instruction take their place. Commonly called *latency hiding*, this concepts allows the efficient use of computing resources without relying on sophisticated control units. However, the workload has to be sufficiently parallelizable to occupy most of the PEs.

[1] based on Calatrava Moreno and Auzinger 2013, Section II.

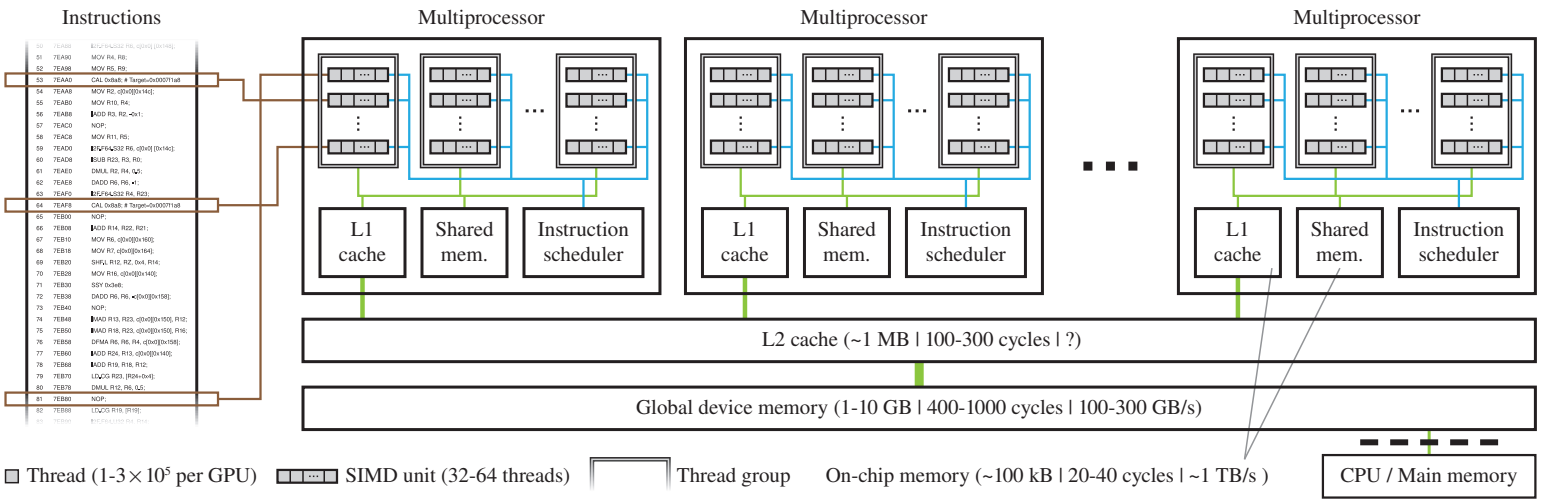


Figure 2.8: Hardware architecture of a modern Graphics Processing Unit. The program instructions (left) are executed in parallel by the many SIMD units of the device. The threads of each SIMD unit are implicitly synchronized since the instructions are issued sequentially to all participating threads of the same unit. The SIMD units are assembled into programmer-specified *thread groups*, which are assigned part of the on-chip memory of each multiprocessor. Used as a scratchpad, this memory allows the efficient sharing of local data or programmer-guided temporal data reuse. Each multiprocessor uses the remaining on-chip memory as L1 cache and interfaces with the global device memory via an L2 cache. An estimate of the specification of each memory type is given as (size | latency | bandwidth) for recent GPU models. Note that this figure provides a high-level overview and does not cover all details of the hardware architecture. Taken from (Calatrava Moreno and Auzinger 2013).

To maximize the number of actual computations that are performed in each clock cycle, instructions are executed in a Single Instruction Multiple Data (SIMD) fashion. Instead of letting each PE compute only a single computation for each instruction, it operates on multiple data elements simultaneously. This reduces the number of required control logic units on the chip estate and provides about an order of magnitude higher compute performance than comparable CPUs. As a downside, however, this architecture requires all lanes of a SIMD unit to perform the same operation. If part of a SIMD unit does not participate in a given instruction, it has to idle. Often referred to as *SIMD divergence*, this property can cause significant performance degradation if the conditionals cause different parts of the same SIMD unit to follow different code paths. As only one instruction can be executed for the whole unit at once, all code paths have to serialized and inactive parts idle.

Furthermore, GPUs sacrifice large on-chip caches for a large bandwidth to device memory, which is about an order of magnitude larger, when compared to CPUs. A small on-chip L1 cache is used both as user-controlled scratchpad to enable data reuse and as L1 cache to distribute the data from read operations to the elements of the SIMD units. A device-wide L2 cache handles both read and write memory transfers. Since memory transfers are issued per SIMD unit, each memory transaction transfers a whole cache line, which contains one data element for each unit element. Thus, it is paramount to ensure coalesced memory transfers, where a read or write instruction of a Processing Element (PE) loads or stores a continuous segment of data.

Given these observations, our parallelization strategies took the following guidelines into consideration:

Device occupancy The workload has to exhibit sufficient parallelism to saturate the thousands of simultaneously active threads on the GPU. Through appropriate load balancing, each compute unit should receive comparable amounts of work to ensure saturation throughout the duration of the whole computation.

Code divergence A two-level parallelization has to ensure that intra-SIMD branching is minimized, while the workload is balanced among the SIMD units themselves.

Memory layout Data elements that are processed in parallel in the same instruction should receive input and store output as continuous segments in memory. This can require intermediate sorting and compacting operations to map the output of one processing stage to the required input layout of a subsequent stage.

Data reuse To reduce the required amount of memory transfers of bandwidth-bound processing stages, we make use of the on-chip scratchpad to reuse intermediate results.

Resource management The number of registers and the amount of the scratchpad memory that each SIMD unit utilizes is determined at program compilation. Since both resources are limited, excessive use of one or both should be avoided as it

could result in a low number of SIMD units that can be run concurrently. This would lead to a device underutilization and reduced memory latency hiding.

Especially in the section on exact visibility in Chapter 5, an application of these strategies is given.

CHAPTER 3

Curved Surface Reformation

In this chapter, we present a sample-based method to resolve the visibility of multiple cut surfaces through three-dimensional volume data. By automatically creating adaptive cutaways, the occlusion of relevant parts of each surface is minimized. As an application, a medical visualization framework for blood-vessel inspection is presented.

3.1 Motivation

Vessel diseases, such as arteriosclerosis or pulmonary embolisms, are becoming an increasing concern in our aging society. Thus, the analysis of blood vessels for their diagnosis and treatment are important research fields of radiology. Angiographic interventions such as stenting, balloon dilatation, or bypass surgery, need to be planned with care and precision, due to their major impact on the patient's health state. To optimally decide on the therapeutic procedure, special diagnostic methods are required. They assess the health state of vessels and answer clinically relevant questions, e.g., if blood is partially or entirely hindered from flowing through a vessel by calcifications or soft plaque.

The gold standard for angiographic imaging is Digital Subtraction Angiography (DSA), a non-tomographic procedure. However, with the capabilities of modern medical scanners, less invasive, tomographic methods, like Computed Tomography Angiography (CTA) or Magnetic Resonance Angiography (MRA), are becoming increasingly popular. Both methods require the injection of a contrast agent into the patient's vasculature before the actual scan, in order to enhance the contrast in the resulting data set.

Confronted with a tomographic data set, special visualization techniques are required to assess possible vessel pathologies. Among these methods are Curved Planar Reformation (CPR) and Centerline Reformation (CR), which create a curved cut through a vessel along its central axis (*interlink*) to visualize its interior (*lumen*). Through cut surface rotation around the centerline of the vessel, they enable the identification of relevant

blood flow obstructions or pathologies at the vessel walls. Curved Planar Reformation (CPR) has been initially designed for the visual analysis of peripheral arterial occlusive diseases, where the vessels are predominately oriented along the human lower extremities. Visualizing spatially arbitrarily oriented vessels leads to artifacts with this method in a majority of cases. Centerline Reformation (CR) addresses this limitation by employing wave-front propagation for the lumen visualization. However, both methods operate in 2D image space, thus suffering from a premature discretization of the vessel geometry, like decreased precision of centerline location and loss of visibility information.

With Curved Surface Reformation (CSR) we propose a method designed to rectify these shortcomings (see Table 3.1). The main contributions of our work are:

- A method to generate the cut surface along a vessel centerline fully in 3D, thus avoiding limitations of previous 2D methods.
- The correct handling of occlusions of different parts of the vasculature by employing a cost function on the cut surfaces.
- An automatic generation of cutaways on the cut surfaces to reveal as much of the vessel tree as possible.
- A Level of Detail (LOD) approach to smoothly extend the cut surface into the surrounding tissue.
- The ability to freely rotate, translate and zoom into the vasculature at interactive frame rates.
- An outline of the corresponding rendering pipeline and its implementation as well as feedback by domain experts.

The remainder of the chapter is structured as follows. In Section 3.2, related work from medical visualization is presented. Section 3.3 describes our proposed method in detail from both theoretical and practical points of view. In Section 3.4 results are demonstrated and discussed. An evaluation is given in Section 3.5 and the chapter is concluded with a discussion in Section 3.6.

3.2 Related Work in Visualization

The method proposed in this work extends several aspects of CPR and CR, which are shortly described in this section with other related work. An in-depth survey can also be found in the doctoral thesis of Mistelbauer (2013).

Curved Planar Reformation (CPR). Planar cuts through a data set are intended to provide more insight into otherwise obscured internal structures. Kanitsar et al. (2002) describe Curved Planar Reformation (CPR) as a curved cut through a data set along the centerline of a single vessel. They present three different types of CPR

	CPR	**CR**	**CSR**
Lumen Rendering	Pixel-Based	Pixel-Based	Ray Casting
Vessel Orientation	Restricted[1]	Unrestricted	Unrestricted
View Direction	Restricted[2]	Restricted[2]	Unrestricted
Visibility	None	Fair[3]	Good
Interactive	Yes	Fair	Yes

[1]vessels should run along one major direction
[2]rotation around one axis
[3]trade-off between visibility of lumen and surrounding tissue

Table 3.1: Aspects and improvements of CSR with respect to CPR and CR. Our method retains all positive properties of the most relevant state-of-the-art techniques and extends several aspects significantly.

(projected, stretched, and straightened) with different geometric properties. The projected CPR is neither isometric nor conformal, whereas the stretched CPR is conformal. The straightened CPR is isometric and perspective occlusions due to rotations cannot occur, but the spatial perception of the vasculature as a whole is reduced. Multipath Curved Planar Reformation (mpCPR), proposed by Kanitsar et al. (2006) and evaluated by Roos et al. (2007), extends CPR to multiple vessels and was initially designed to investigate peripheral arterial occlusive diseases. Spiral CPR, also described by Kanitsar et al. (2006), shows the interior of a vessel by flattening it along a spiral. This approach preserves isometry, but lacks spatial context. Kanitsar et al. (2003) propose untangled CPR to avoid occlusions when projecting the vessel tree by using spatial deformations. Mistelbauer et al. (2012) propose Centerline Reformation (CR) as a technique to visualize the lumen of arbitrarily oriented vessels. However, there is a trade-off between the visibility of the vessels and the displayed size of the surrounding tissue. In contrast to CPR and CR, our approach computes the lumen entirely in 3D without projecting the vessel centerlines into 2D image space and remedies the trade-off of CR. Mistelbauer et al. (2013) combine CPRs of several view directions into a single static image by aggregating certain features circularly along vessels. The vessels are shown straightened and rotation operations are unnecessary. Lampe et al. (2009) introduce a more general curve-centric reformation that allows transforming the space around a curve. Their method produces visualizations similar to planar reformations used for virtual endoscopy. Williams et al. (2008) present a technique for colon visualization based on a combination of CPR and Direct Volume Rendering (DVR). Jianu et al. (2012) explore the brain connectivity by projecting 3D fiber tracts into 2D and subsequently cluster them, with each cluster having one centroid and many non-centroid tracts. They account for visibility by depth sorting the centroid tracts according to the depth of the center of the corresponding 3D segments. Lee and Rasch (2006) introduce a tangential curved planar reformation by sweeping the Frenet frame along the projected centerline of a vessel. This creates

artifact-free visualizations of the vessel lumen, but not of the surrounding tissue. Saroul et al. (2003) propose a technique for flattening free-form surfaces of anatomical medical structures that minimizes distortion along a user-specified curve or direction (Saroul et al. 2006). The surface is spanned between boundary curves, defined by users interactively placing markers within the 3D volume. CSR creates a view-dependent cut surface of the vasculature without any user interaction.

Context Visualizations. Straka et al. (2004) propose the *VesselGlyph* to establish focus-and-context rendering for CPR and mpCPR. It provides the possibility to place vessels within a context visualization like Maximum Intensity Projection (MIP) or DVR. Viola et al. (2006) discuss a method for retaining context while refocusing on different structures of interest in a volume data set. This is also supported by our technique. DeCarlo et al. (2003) present *suggestive contours* as the non-photorealistic rendering of lines in order to provide visual cues of a shape. Sander et al. (2000) describe silhouette clipping as an approach to produce faster qualitatively equal silhouettes for a coarser mesh than with the high-resolution one.

Cutaways. Diepstraten et al. (2003) present a set of rules for rendering interactive cutaways of polygonal data. Burns and Finkelstein (2008) describe view-dependent cutaways for polygonal scenes. They distinguish between two different cutaway approaches: cutout and breakaway. Whereas in the former one half spaces are intersected, the latter is realized by a single hole in the outer object. They state that a jittered cutting edge leads to a higher abstraction level. Li et al. (2007) present a system for creating cutaways of complex objects in real-time. They apply visual styles from conventional illustrations such as various cutting types (box, tube, or window cuts), visual cues (edge shadows and shading) and inset cuts that cut deeper layers of an object with a smaller extent. Sigg et al. (2012) propose a degree-of-interest function for interactive specification of important features instead of manually placing them. In an iterative user-centric optimization process, cutaway primitives (cuboids, spheres, or cylinders) are optimally placed in order to allow a clear view of the desired object in the context of polygonal meshes and volumetric data. They use simulated annealing for optimally placing these primitives and apply their technique to polygonal meshes as well as volumetric data. McInerney and Crawford (2010) present a paint roller as an interaction metaphor for specifying the shape of the cutaway manually, while retaining ribbons of the occluding object. Burns et al. (2007) use an importance function for generating volumetric contextual cutaways for medical visualization. They obtain a smooth transition between the occluder and the important object. Lidal et al. (2012) give design guidelines with the aim to enhance shape and depth of the focused object in relation to its context. McGuffin et al. (2003) use volume deformations and a set of interactive widgets to offer users the possibility to virtually browse through different layers of medical volume data. CSR will automatically create cutaways (see Figure 3.6d) to reveal occluded vessels by means of a cost function.

Volume Clipping. Konrad-Verse et al. (2004) describe a virtual resection tool for organs employing user-placed meshes. After the user initially paints a stroke on the desired organ, a virtual cutting plane is computed via a principal component analysis. Additionally, the plane can be deformed in order to include or exclude objects from the cutting. Birkeland et al. (2012) describe an illustrative clipping surface by means of physical properties of elastic membranes. It can serve as an illustrative slice with local DVR confined to its vicinity. User adjustment is enabled by introducing additional force fields. To avoid cutting near objects, the surface adapts to salient features of the underlying data. Weiskopf et al. (2003) describe interactive clipping of convex and concave objects from a volume, while accounting for proper volume shading. They account for proper volume shading together with clipping by shading the cutting surface accordingly. Wang and Kaufman (1995) propose a method for volumetric sculpting. They apply various operations on the volume at voxel level. Zhang et al. (2007) present a real-time clipping strategy for visualizing data inside a volume. They compare several classifications for volume rendering with ray casting. For further information considering physically based deformable models, we refer to the survey by Nealen et al. (2006). In our approach, the volumetric data is cut by a curved surface.

Curve Simplification. In our technique, we simplify centerline curves to generate a coarse representation of the vessel tree to infer the large-scale behavior of the cut surfaces. As an old problem from computational geometry, it was already investigated by Imai and Iri (1986; 1988) for polygonal curves in the late 80s. The most commonly used algorithm was developed by Ramer (1972) and Douglas and Peucker (1973). Recent developments consider an improved distance metric (Gudmundsson et al. 2007) or utilize Bézier curves (Chuon et al. 2011) for simplification.

3.3 Curved Surface Reformation

In the field of radiology, the diagnostic visualization of vascular structures plays a vital part in the investigation of certain vessel pathologies. Among occlusive diseases, stenoses are of particular interest. They can be caused by calcifications or soft plaque on vessel walls. The manual analysis of the raw medical imaging data of vessels is very time consuming, and the use of medical visualization can improve the efficiency dramatically. Visualization methods usually require additional information on the imaging data, such as the locations of the vessel centerlines. These are modeled by radiological assistants in their daily clinical routine, in which they are assisted by semi-automatic detection methods such as the technique of Kanitsar et al. (2001). In the context of our visualization technique, centerline data is treated as input that was generated in a preprocessing step. Specialized visualization algorithms are then used for inspection of regions of interest and detection of pathologies.

General purpose methods can produce misleading visualizations. An example, when using MIP, is that vessels, which have concentric calcifications on their walls, incorrectly

appear as completely occluded. To remedy this, planar reformations create a cut through the tissues or, as it is the case for vessel reformation, a cut along the vessels to visualize their interior. These approaches provide a good overview of the vessel lumen and enable the specialist to deliver a qualified judgment on the severity of pathologies.

Different vessel reformation methods make various assumptions on the vessel geometry. Curved Planar Reformation (CPR) and its extension for multiple vessels, Multipath Curved Planar Reformation (mpCPR) (Kanitsar et al. 2006), have been designed for the inspection of peripheral arterial occlusive diseases, where vessels strongly follow one principal direction. For display, the vessel is aligned with the up-axis $\boldsymbol{u}$, and the view direction is taken orthogonal to it. The cut surface is then spanned by the centerline and the direction that is orthogonal to both $\boldsymbol{u}$ and the view direction. This entails two major restrictions: the view direction can only rotate around the up-axis, and the cut surface degenerates if the vessel direction becomes orthogonal to $\boldsymbol{u}$. The latter case leads to excessive vessel thinning as shown in Figure 3.7a. A possible workaround to alleviate this effect is the stretching or straightening of the vessels. However, this interferes with the spatial perception of the displayed vessels, which is usually not desired by domain experts. Furthermore, the initial projection of the centerline to 2D discards the visibility information, and overlapping vessels cannot be kept apart.

Centerline Reformation (CR) (Mistelbauer et al. 2012) removes some of these drawbacks by using wave-front propagation on the projected vessel data. Although this enables the depiction of the lumen of arbitrarily oriented vessels, it still operates in 2D image space. This leads to a significant loss of visibility information due to the necessary projection. The cut surface is created by spreading the vessel locations into their surroundings. The visibility then has to be reconstructed by walking along the projected vessel tree. This leads not only to performance issues, but also requires the use of thresholds. They effectively determine the method's ability to either display the surrounding tissue or to correctly resolve the visibility. Figure 3.7b and Figure 3.8b provide an illustration of this trade-off.

To address these issues, we propose Curved Surface Reformation (CSR) to perform the lumen rendering entirely in 3D space. This new method allows the determination of the correct visibility of the various cut surfaces along the vessel tree. It also enables the extension of these surfaces into the surrounding tissue in a well-defined manner. Furthermore, we offer the possibility to use common interactions such as zooming, panning, or rotating around arbitrary axes in 3D for the first time in the field of vessel reformations. Since our method employs ray casting, it is trivially parallelizable by assigning each ray to a thread. This maps well to parallel hardware architectures and our Compute Unified Device Architecture (CUDA) implementation that runs at interactive frame rates on modern Graphics Processing Units (GPUs).

In Section 3.3.1 the theory of our method is developed in a continuous setting. Its application on discrete input data is outlined in Section 3.3.2. Sections 3.3.3 and 3.3.4 discuss the resulting rendering pipeline (see Figure 3.3) and its implementation.

3.3.1 Theory

To render the lumen of a vessel, we employ ray casting of a surface given by a cut through the vessel along its centerline. As we treat arbitrarily oriented vessels and do not require a fixed camera position, the cut surface is generated dynamically, with its geometry depending on both the view direction and the orientation of the corresponding vessel.

In this section, we assume as input to our method a set of differentiable 3D curves, referred to as *vessel segments*, which constitute the centerlines of the relevant part of the vasculature for the visualization task. While our method does not require connected segments and is able to handle arbitrary topologies of the segment graph, the typical use case is a connected set of vessel segments with possible branchings at their endpoints. We will refer to this as the *vessel tree*. Information on the radius of the vessel along the centerline is optional and can be used in the visibility computations (see Section 3.3.1). As the geometrical concepts can be presented much clearer in a continuous setting, we take this route and describe the subsequent adaptations to the discrete input data in Section 3.3.2.

Surface Generation

Given the centerline curve $\boldsymbol{c}(t) : [t_0, t_1] \to \mathbb{R}^3$ of a vessel segment and a view direction $\boldsymbol{v}$, the *local (cut) surface* M is given by

$$M(s,t) = \boldsymbol{c}(t) + s\,\boldsymbol{v} \times \boldsymbol{c}'(t) \quad \text{with } t \in [t_0, t_1] \text{ and } s \in \mathbb{R}. \tag{3.1}$$

This definition is motivated by the requirements of having the local surface containing the centerline and being as perpendicular to the view direction as possible. CPR and mpCPR, in comparison, use a fixed frame in the form of $\boldsymbol{c}(t) + s\,\boldsymbol{v} \times \boldsymbol{u}$, where $\boldsymbol{u}$ denotes the up-axis, and the view vector is constrained by $\boldsymbol{v} \cdot \boldsymbol{u} = 0$.

To show a vessel cross section at the endpoints, we add two half-planes that are perpendicular to the view direction, i.e., for the start $\boldsymbol{c}(t_0)$ and endpoint $\boldsymbol{c}(t_1)$, the half-planes are given by

$$\boldsymbol{c}(t_0) + s\,\boldsymbol{v} \times \boldsymbol{c}'(t_0) + t\,\boldsymbol{v} \times (\boldsymbol{v} \times \boldsymbol{c}'(t_0)) \quad \text{and} \quad \boldsymbol{c}(t_1) + s\,\boldsymbol{v} \times \boldsymbol{c}'(t_1) - t\,\boldsymbol{v} \times (\boldsymbol{v} \times \boldsymbol{c}'(t_1))$$

for $s \in \mathbb{R}$ and $t \in \mathbb{R}_+$. In the special case of an alignment of the centerline and the view direction ($\boldsymbol{v} \times \boldsymbol{c}'(t) = \boldsymbol{0}$), a half-plane with a boundary perpendicular to the last non-aligned centerline direction has to be added. If the centerline is parallel to the view direction at one of its endpoints, we add a plane perpendicular to the view direction in this point. This ensures that a projection of the resulting local surface covers the whole view plane at least once and that the vessel cross section is visible in case of a collinearity with the view direction.

In a sufficiently small neighborhood of the centerline, this construction provides the correct cut surface for the corresponding vessel segment. For a finite extent of the vessel, however, this naive approach exhibits various deficiencies. Our proposed method on

how to rectify these issues to enable a full 3D reformation is presented in the following sections.

Main Challenges

The local surface M, as given in Equation 3.1, depends on both the view direction $\boldsymbol{v}$ and the local geometry of the centerline, which is specified by its tangent vector $\boldsymbol{c}'(t)$. While this is appropriate for a small neighborhood of the centerline, this definition shows two types of undesired behavior when moving away from it. As the vessel lumen near the centerline is the main point of interest, it should be visible whenever it is not directly occluded either by the same centerline or different vessel segments. This behavior is not exhibited by M, as only the front-most parts of the whole vessel tree are visible. The second problem is that small local details or noise are amplified at the outer regions of the surface, which leads to self-occlusions and depth discontinuities when viewed along the view direction $\boldsymbol{v}$ (see Figure 3.2). A solution to the first problem is presented in Section 3.3.1 and to the second one in Section 3.3.1.

Cost Function

To enable the inspection of as many vessels as possible in the same view, it is beneficial to reveal the local surface of all those vessel segments that are not located behind other vascular structures in the current view. Hence, we create cutaways on the local surfaces of vessel segments closer to the viewer based on two simple principles:

1. The closer a point on a centerline is located to the viewer, the more relevant is the associated part of the local surface.
2. The closer a point on a local surface is located to its centerline, the more relevant it is.

With the use of spatial cost functions for all vessel segments, it is possible to mimic this behavior. For each centerline c_i of the vessel tree, we define a cost function

$$C_i(\boldsymbol{x}) = C_{\parallel}(\boldsymbol{x}) + \lambda C_{i,\perp}(\boldsymbol{x}), \tag{3.2}$$

where λ denotes the weighting between the component parallel to the view vector $\boldsymbol{v}$, i.e., $C_{\parallel}(\boldsymbol{x}) = \frac{\boldsymbol{x}\cdot\boldsymbol{v}}{\|\boldsymbol{v}\|}$, and the one normal to it. The latter is given as the distance $dist_{v,\perp}(\boldsymbol{x}, c_i)$ between $\boldsymbol{x}$ and the image of curve c_i when both are projected onto a plane P normal to $\boldsymbol{v}$. The intuitive explanation of this procedure is that the parallel cost component penalizes the depth of surfaces while the normal component penalizes the distance from the respective centerline.

By evaluating the cost function on all local surfaces and projecting these values onto a plane P normal to $\boldsymbol{v}$, we obtain several cost values per point on P. Actually, each local surface contributes one or more cost values to each point on P, as they cover the full extent of P due to the nature of their construction. See Figure 3.1c for a local surface

that contributes six cost values for each point on P in the center region of the trefoil knot. Another illustration can be found in Figure 3.5d. At each point we now select the smallest cost value and mark the corresponding surface location as visible. As a result, for each point on the view plane, the visible local surface is selected and the global visibility is resolved (see Figure 3.1). Ambiguities that arise from several identical cost values at the same point of P can be neglected almost always, as these are part of one-dimensional boundaries where the visibility changes between different centerlines. In very rare cases it is possible that whole local surface regions contribute the same cost values and that artifacts similar to z-fighting arise. As this behavior only occurs for specific view directions (we conjecture that these form a one-dimensional subset of the two-dimensional space of all view directions), we feel safe to ignore them as we did not encounter them in any of our experiments.

The parameter λ in the cost function encodes the preference for broader or thinner surrounding surfaces around vessels. A large λ heavily penalizes the projected distance, and thus the cutaways of occluded vessels reach close to the visible centerline. The contrary case of a small λ leads to a more liberal occlusion of vessels behind the current centerline.

A minor modification can be applied to the cost function to guarantee the preservation of the full extent of vessels, such that cutaways do not cut into the vessel lumen of a visible centerline. This can occur in the case of two centerlines with a small depth difference, as the cost function indicates a preference for selecting the occluded vessel very near to the centerline of the occluding vessel. To avoid this situation, the normal component of the cost function has to be modified to assign no distance cost to regions around the centerline within a certain vessel radius. This can be considered as placing a strip with possibly varying vessel radius around each centerline, in which the normal cost component of the associated cost function is zero. As a consequence, a vessel behind another centerline will always have a higher cost as both the depth and distance component contribute to its occlusion. Note that for very dense vascular structures, a large number of cutaways is generated, and the local surface of most visible vessels will not exceed the extent of the vessel itself. In this case, our method would resemble dense line rendering and due to conceptual similarities, only slight modifications would ensure convergence to the illustrative method of Everts et al. (2009).

Our procedure resolves the visibility of the local surfaces of the vessel tree and thus remedies the first problem with the initial definition of the local surfaces. In the following section, the second challenge as described in Section 3.3.1 is tackled.

Centerline Simplification

Although the local surface M, as defined in Equation 3.1, is geometrically appropriate in a neighborhood of its respective centerline, globally it exhibits an undesired behavior (see Figure 3.2). This is caused by the use of centerline derivations in the definition of M, as small geometrical details along the vessel get amplified with increasing distance from

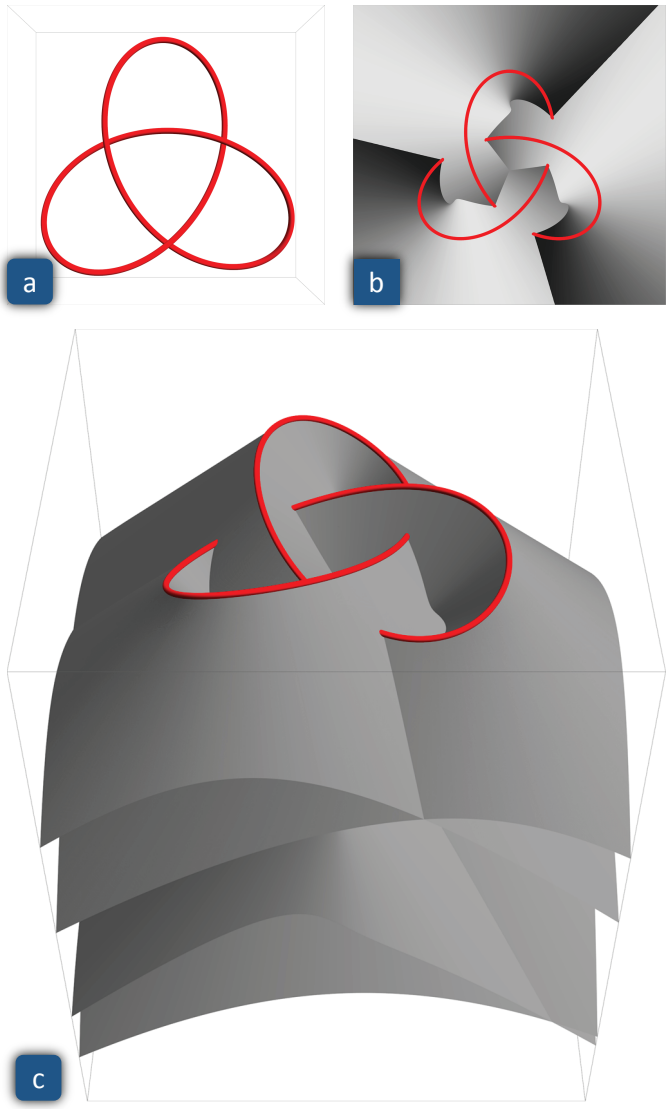


Figure 3.1: Illustration of the visibility resolution using a cost function. (a) shows a synthetic vessel input in the form of a trefoil knot. We use a cost function as given by Equation 3.2 defined on the generated local surface to determine its visible points, which are characterized by the lowest cost value along the view direction. The cost function for a frontal view of (a) is visualized in (c) by encoding the *cost value* as the depth of the local surface. Note that each vertical view ray intersects the local surface several times, and only the topmost intersection points constitute the visible surface. The final result in form of a depth image is given in (b), where the grayscale encodes the depth of the local surface.

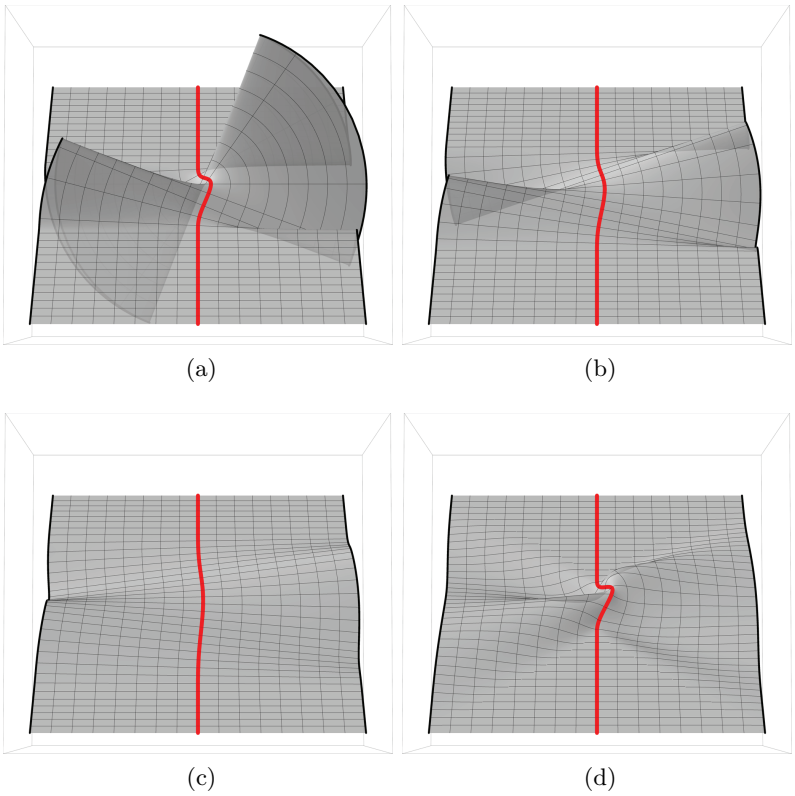


Figure 3.2: Illustration of the instability of the local surface with respect to small perturbations of the centerline (red). (a) exhibits severe distortions by a small scale detail of the centerline leading to multiple self occlusions of the local surface. These can be drastically reduced by progressive smoothing of the centerline as shown in (b) and (c). Our proposed LOD method uses coarser and smoother representations of the centerline for increasingly distant parts of the surface. This eliminates the undesired self-occlusions but preserves the detailed vessel geometry close to the centerline (d).

the centerline. Generally, this leads to a multitude of undesired self-occlusions of the local surface when viewed along the view direction $\boldsymbol{v}$.

In order to combat this behavior, we propose a LOD approach based on centerline simplification. While small centerline details have to be accounted for, in order to enable a faithful representation of the vessel geometry close to it, only the large-scale characteristics of the centerline should govern the surface behavior in the surrounding volume. It would be possible to develop a theory of a continuous simplification of the centerline, yielding infinitely many LODs. However, we employ just a finite number of discrete levels to perform the simplification, which is justified by the discrete nature of the actual input data and the fact that the continuous formulation would not aid further understanding.

For a fixed number $(N+1)$ of LODs, each centerline is simplified to obtain N increasingly smoothed versions of it, where the initial curve is preserved in level 0. We specify the actual simplification method for discrete input data in Section 3.3.2.

The visibility between the local surfaces of the smoothed versions of a centerline is again resolved with the help of the cost function by evaluating it on the local surfaces of relevant LODs. Depending on the projected distance d between the centerline of a vessel segment and a point on the view plane, we determine the two most relevant LODs k_0 and k_1 by using a selection function $f(d)$ to obtain

$$k_0 = \min(N, \lfloor f(d) \rfloor) \quad \text{and} \quad k_1 = \min(N, \lfloor f(d) + 1 \rfloor).$$

To concentrate the initial LODs around the centerlines we use a sublinear selection function; in our implementation of the form $f(d) \propto \sqrt{d}N$.

As described in Section 3.3.1, we project the cost values of the two surfaces onto the view plane and, for each LOD separately, we chose the surface locations $\boldsymbol{i}_0$ and $\boldsymbol{i}_1$ with minimal cost for each point on the view plane. The final *cut surface* location $\boldsymbol{i}$ is then given as a linear blend of the two LODs, i.e.,

$$\boldsymbol{i} = (k_1 - f(d))\boldsymbol{i}_0 + (f(d) - k_0)\boldsymbol{i}_1.$$

This gives a smooth progression between the LODs and a gradual simplification of the local surface as a function of distance d. Note that beyond the last LOD, no blending is performed, since $k_0 = k_1 = N$. A depiction of the result of this method is given in Figure 3.5e.

This concludes the theory section for our proposed method. Before describing the actual rendering pipeline, we outline the necessary adaptation of the aforementioned methods to handle discrete input.

3.3.2 Discrete Geometry

The theory in Section 3.3.1 was developed in a continuous setting, which does not reflect actual computation hardware. This section covers the required discretization steps necessary for numerical processing.

Surface Generation

Assuming that a vessel centerline is given as a piecewise linear curve, the corresponding local surface consists of planar pieces, namely line stripes, connecting triangles and half-planes at the endpoints. In this section, we give a short description on how to generate such a local surface.

For a line segment of the vessel tree given by the points $\boldsymbol{l}_0$ and $\boldsymbol{l}_1$, a line stripe is generated by extruding the line segment in direction $\boldsymbol{v} \times (\boldsymbol{l}_1 - \boldsymbol{l}_0)$ (see Figure 3.5a). As this direction is orthogonal to the view direction $\boldsymbol{v}$, the stripe is located between the depths $d_0 = \boldsymbol{v} \cdot \boldsymbol{l}_0$ and $d_1 = \boldsymbol{v} \cdot \boldsymbol{l}_1$ along the view direction.

If two consecutive line segments L_0 and L_1, given by the three points $\boldsymbol{l}_0$, $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, are not collinear when projected onto the view plane, a gap would arise in the projected local surface in case only line stripes were used. To avoid this, we use a triangle that originates in the connection point $\boldsymbol{l}_1$. The triangle edges extend along the boundaries B_0 and B_1 of the line stripes given by

$$B_0 = \{\boldsymbol{l}_1 + s\,\boldsymbol{v} \times (\boldsymbol{l}_1 - \boldsymbol{l}_0)\} \qquad \text{and} \qquad B_1 = \{\boldsymbol{l}_1 + s\,\boldsymbol{v} \times (\boldsymbol{l}_2 - \boldsymbol{l}_1)\}\,.$$

This ensures that the triangle fills the gap between B_0 and B_1. As such a gap only arises on one side (when projected onto the view plane), we choose either $s \in \mathbb{R}_+$ or $s \in \mathbb{R}_-$.

At the start and end point $\boldsymbol{l}_0$ (resp. $\boldsymbol{l}_n$) of a vessel segment, we add half-planes H_0 (resp. H_n) to complete the local surface. These are defined as

$$H_0 = \{\boldsymbol{l}_0 + s\,\boldsymbol{v} \times (\boldsymbol{l}_1 - \boldsymbol{l}_0) + t\,\boldsymbol{v} \times (\boldsymbol{v} \times (\boldsymbol{l}_1 - \boldsymbol{l}_0))\}$$

and

$$H_n = \{\boldsymbol{l}_n + s\,\boldsymbol{v} \times (\boldsymbol{l}_n - \boldsymbol{l}_{n-1}) - t\,\boldsymbol{v} \times (\boldsymbol{v} \times (\boldsymbol{l}_n - \boldsymbol{l}_{n-1}))\}$$

with $s \in \mathbb{R}$ and $t \in \mathbb{R}_+$.

If a line segment L is collinear with the view direction, we add an additional half-plane using the definition of H_n above, but substituting the points $\boldsymbol{l}_{n-1}$ and $\boldsymbol{l}_n$ by the endpoints of the last non-collinear line segment before L along the centerline. If the first or last line segment of a centerline is collinear, we create an additional plane normal to $\boldsymbol{v}$ that contains the corresponding endpoint.

These constructions ensure that the piecewise planar surface of each centerline covers the whole view plane when projected onto it as illustrated in Figure 3.5b. As we employ ray casting to evaluate the cost function on this surface, we can use simple geometrical intersection tests between the rays and the unbounded polygons. The cost function of Section 3.3.1 can be directly used for this purpose.

Centerline Simplification

The LODs of the vessel tree are computed in a preprocessing step. We employ progressive filtering with spline functions to generate coarser representations of the individual vessel segments. The centerline of a given LOD is approximated by a cubic B-spline and resampled to generate the next LOD. The performance of this simple approach is sufficient, and the quality gains by employing more sophisticated methods such as the curve simplification algorithm by Chuon et al. (2011) are negligible. Due to the absence of fine details in higher LODs, it is possible to faithfully represent the vessel segments with less line segments. In our implementation, we pursue an aggressive pruning as we drop half of the points for each successive LOD. For a sufficiently dense initial sampling of centerlines as in our input, we did not observe any artifacts resulting from this optimization, which drastically improves runtime.

As described in Section 3.3.1, our LOD approach uses the selection function $f(d)$ depending on the projected distance d of the ray from the centerline. This value can be easily determined by computing the distance between the ray and the line segments of the initial centerline, i.e., the unsmoothed LOD 0.

3.3.3 Rendering

This section presents the rendering pipeline of CSR as outlined in Figure 3.3. It takes a vessel tree, consisting of line segments, and a volume representation of the surrounding tissue as input and outputs an image of the cut surfaces along the vessel centerlines. As we employ ray casting at several stages of the pipeline, various buffers in the size of the final output image are used to store intermediate values. A sequence of such intermediate results is given in Figure 3.5. In the exceedingly rare case of z-fighting artifacts (see Section 3.3.1), we prevent their aggravation with temporal flickering by maintaining a strict rendering order of the input data. As a preprocessing step, the LODs of the vessel tree are computed as described in Section 3.3.2.

LOD Estimation

This stage generates a distance buffer that stores the projected distance of each image location to the nearest centerline of the vessel tree. This data is used as the argument d for the LOD selection function $f(d)$ as described in Section 3.3.1.

Depth Computation

Using the distance information of the previous stage, we determine the two relevant LODs using the method of Section 3.3.1. By using ray casting, we collect the cost values for all intersection points with local surfaces along the ray. This is done separately for each pair of LODs, and we retain the intersection points with the minimal cost for each pixel. The final intersection point is a weighted blend between the respective LOD intersections to ensure smooth progressions from one LOD to the next. This stage outputs the depth of the final intersection point to a depth buffer.

Depth Filtering

To remove aliasing artifacts that arise from the undersampling of a high-frequency depth signal caused by fine geometric details of the vasculature, the depth image is convolved with a smoothing filter. This is necessary when sampling cut surfaces, which extend from centerlines that are nearly collinear with the view direction, since their planar parts shrink to sub-pixel sizes. We use a Gaussian filter kernel with standard deviation σ for this task and set its full width at half maximum (FWHM) in frequency space to half the Nyquist frequency of the output image, i.e., $\sigma = 2\sqrt{2 \ln 2}/\pi$ pixel. The discrete version is cut off at 3σ, which gives a kernel size of 7×7. The output makes up the depth values of the desired cut surface (see Figure 3.5c).

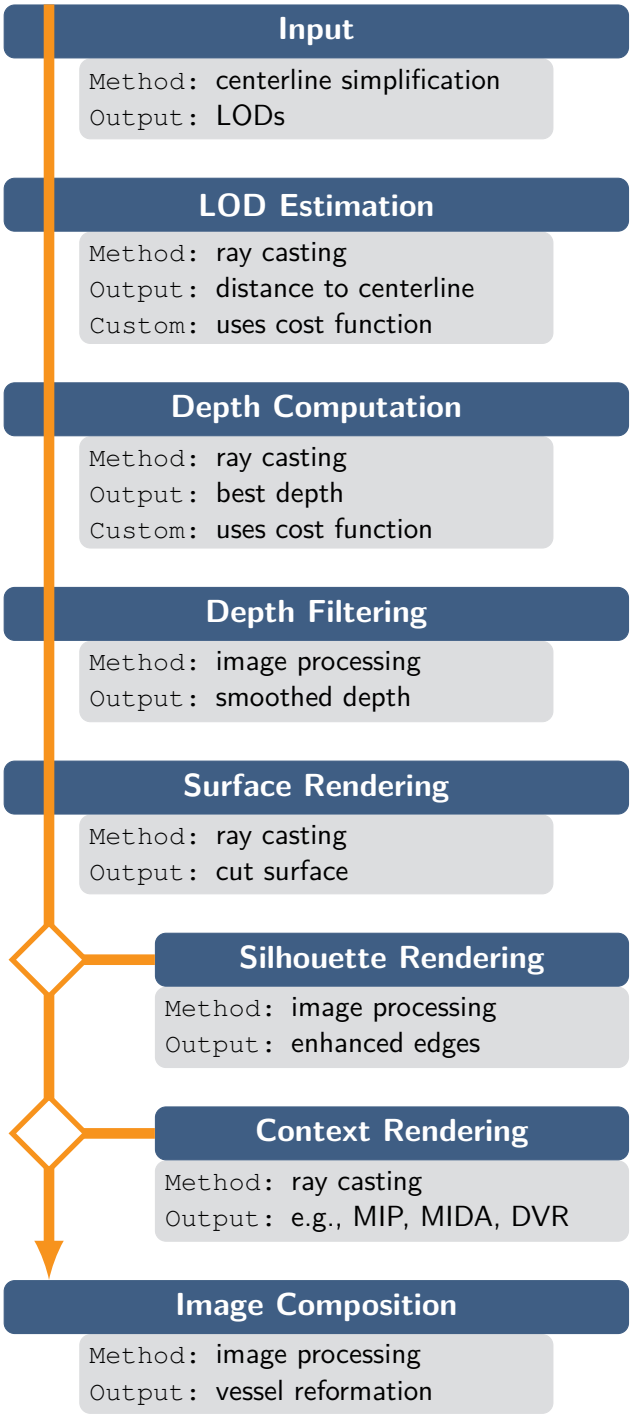


Figure 3.3: Rendering pipeline of Curved Surface Reformation (CSR). The first entry of every stage denotes the used method and the second item gives the output. The custom entry indicates the stages where the cost function is used.

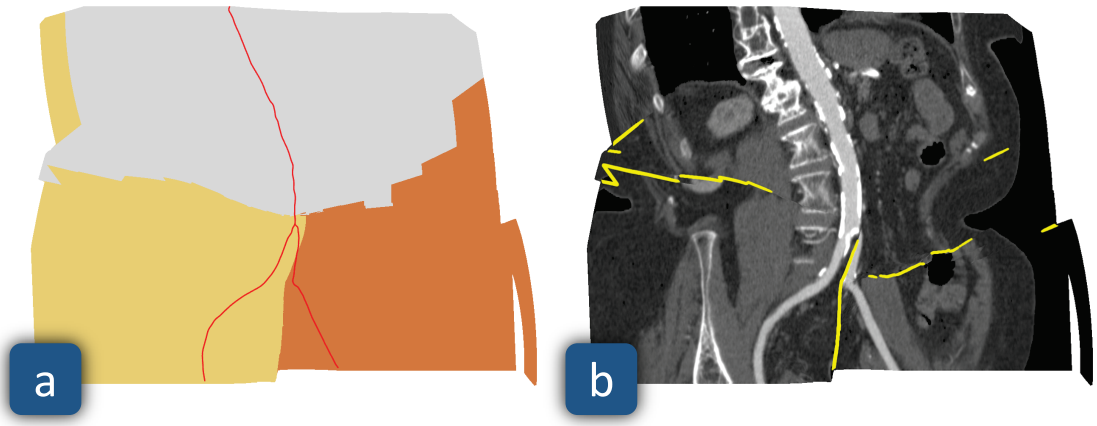


Figure 3.4: Silhouette rendering. (a) shows the color coded influence zones of the three vessel segments. Small depth discontinuities at the zones' borders and large discontinuities anywhere are presented as yellow silhouettes in (b) as an overlay of the cut surface.

If the filter size is significantly increased, the depth discontinuities between the visible cut surface regions of different parts of the vessel tree are smoothed. In the final rendering, this leads to steep 'slopes' between the regions. We do not utilize this fact in our method and leave it just as an observation.

Surface Rendering

A simple lookup into the volumetric intensity data set, which surrounds the vessel centerlines, is performed with the depth values of the cut surface. We use trilinear interpolation—a supersampling methodology—for this task and obtain the intensity values of the visible vasculature's lumen and of the surrounding tissue. Afterwards, we apply a user-specified windowing function to map these values to grayscale colors and opacities.

Silhouette Rendering

After determining the depth of the cut surface and the subsequent filtering step, depth discontinuities still remain. These are, however, intentional, and constitute the switch from the surrounding of one part of the vessel tree to another one. Since these important spatial cues might not be easily perceivable, we improve the three-dimensional visual perception of the cut surfaces by highlighting these discontinuities. This is achieved by modulating a user-specified silhouette color (yellow in our case) with the response of a customized filter function.

Large depth discontinuities are revealed by applying a Sobel edge-detection filter. However, small discontinuities between spatially close vessel segments from different parts of the vasculature are not captured. To identify these, every point of the projected cut surface

is marked with a unique identifier of the corresponding vessel segment (see Figure 3.4a). Small depth discontinuities can only occur along the borders of these influence zones of the vessel segments while branching locations remain unaffected, since they are continuous in depth.

Thus, we amplify the response of the Sobel filter at these borders. We observed that a simple multiplication by a constant factor already yields adequate results and improves the depth perception of vessel occlusions near branching locations. A final smoothing of the combined filter response provides anti-aliasing of the discontinuity lines and gives our final color modulation (see Figure 3.4b). This allows for a better perception of depth discontinuities.

Context Rendering

Due to the fact that the cut surface displays only parts of the volume, the perception of the overview can suffer. To remedy this, we optionally add a final context visualization step to the CSR workflow. Straka et al. (2004) propose a focus+context approach by embedding the vessel lumen (focus) into a volume rendering (context). We render a context behind the regions of cut surfaces at locations where they lie outside the volume. When rotating, the context visualization becomes apparent, and the domain expert can clearly perceive the spatial location of the region of interest within the data set. Figure 3.5f shows such a context visualization, using Maximum Intensity Difference Accumulation (MIDA) as proposed by Bruckner and Gröller (2009). Any other volume rendering technique could serve as context visualization as well.

The context is displayed, whenever a ray hits the volume and the cut surface lies outside the volume at its pixel location or when the windowing function makes the cut surface fully transparent. In the first case, the rays traverse the whole volume, whereas in the second case, the rays start at the depth of the cut surface (see Figures 3.6b-d, 3.5f, 3.7d). The advantage of this context visualization is that it provides additional spatial cues when rotating while still depicting the occlusion-free cut surface. This is especially useful in case of unconstrained rotations, because one might quickly lose the orientation when only viewing the cut surface.

3.3.4 Implementation

Our technique was implemented as part of the AngioVis framework, which is in clinical use at the Vienna General Hospital and the Kaiser-Franz-Josef Hospital. Our extension is not yet in clinical use. The LODs of the vessel tree centerlines are generated on the Central Processing Unit (CPU) as a preprocessing step at startup and regenerated after changes to the vessel tree. The remaining stages of Section 3.3.3 are implemented in CUDA and use Open Graphics Library (OpenGL) just for computing the entry and exit positions of the rays. This allows the whole pipeline to make use of the parallel computing capabilities of current GPUs which leads to an interactive performance of our implementation.

Figure 3.5: Illustration of our rendering pipeline. (a) depicts an input vessel tree consisting of 800 line segments. A detailed view of a single line segment illustrates the variables of Section 3.3.2 with the view direction orthogonal to the plane in which this figure is embedded. The geometry of the local surface of the unsmoothed centerlines is shown in (b) where the planar parts are colored according to their type – dark brown for line stripes, light brown for triangles and blue for the half-planes at the endpoints. The final depth buffer after the depth computation and filtering stages is depicted in (c) and the corresponding cost function is illustrated in (d). (e) gives an overview of the LODs which are alternately colored in red and green. It also shows the part of the final output (f) that is covered by the cut surface whereas the other parts are generated by the context rendering stage.

Figure 3.6: Bifurcation of the human abdominal aorta. (a) shows a 3D direct volume rendering of the data set together with the surface of the vessel tree (red wireframe). The remaining three images present our approach for visualizing the vessels' interior by generating a view-dependent cut surface through the vasculature. Our method features a smooth cut through surrounding tissue (b), correct vessel lumen portrayal even at grazing angles (c) and the automatic placement of cutaways to allow inspections of large parts of the vasculature in a single view (d). The visibility of the vessel tree is correctly resolved for every view direction.

	Figure 3.6	**Figure 3.7**	**Figure 3.8**
Data Size	$512^2 \times 256$	$512^2 \times 1305$	$512^2 \times 575$
LOD Estimation	97.8 ms	272.6 ms	133.0 ms
Depth Computation	13.5 ms	105.6 ms	39.0 ms
Depth Filtering	1.7 ms	1.7 ms	1.7 ms
Surface Rendering	0.9 ms	1.0 ms	1.1 ms
Silhouette Rendering	7.6 ms	7.7 ms	7.7 ms
Context Rendering	96.9 ms	270.7 ms	131.8 ms
Frame Time	218.4 ms	659.3 ms	314.3 ms

Table 3.2: Performance timings in milliseconds per frame of CSR for the data sets presented in the results. The most time consuming operations are the LOD estimation, as every line segment has to be evaluated during this stage, and the context rendering. The depth computation is less expensive and all other operations have negligible temporal cost.

We analyzed the computing times of all individual stages of the CSR rendering pipeline. The outcome is summarized in Table 3.2 with the timings given in milliseconds. The measurements have been taken on an Intel Core i7 with 3.07 GHz with 12 GB system memory and a GeForce 680GTX with 4 GB video memory. The LOD estimation is the most costly step, as each ray is checked with every line segment of the most detailed LOD of the vessel tree in order to compute an accurate distance. The performance of this step could be increased by gradually computing the distance from the coarsest to the finest LOD and updating only relevant distances in each step. The depth computation is less expensive, since only two coarse LODs are tested for most of the rays. The following operations consume nearly constant, insignificant amounts of time. Context rendering takes a considerable amount of time due to the ray marching of MIDA. Nevertheless, even our unoptimized implementation of CSR is capable of interactively rendering the cut surface together with the context visualization. Ray casting could also be substituted by rasterization of the local surface primitives similar to the method of Hoff et al. (1999). However, the performance benefits of such an approach are not guaranteed, since the cost function methodology does not allow the utilization of the hierarchical depth-buffering capabilities of graphics hardware.

3.4 Results

In this section, several results that were obtained with our method are presented. The weighting $\lambda = 10$ in the cost function (see Equation 3.2) is used for all subsequent results. First, we present a small data set to illustrate the capabilities of our technique. With the second and third data set, we compare against existing methods, such as mpCPR and CR, in order to highlight our improvements.

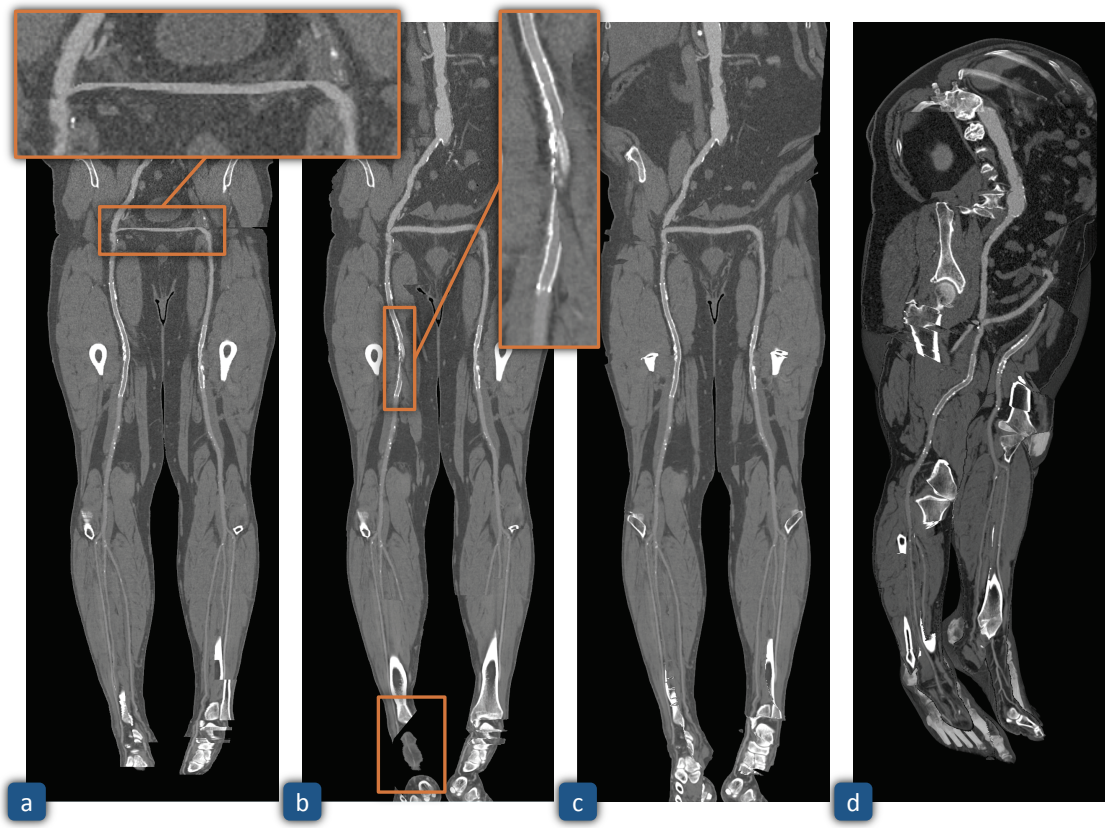


Figure 3.7: A CTA data set of the vasculature of the human lower extremities with a cross-over bypass. (a) shows an mpCPR of the vasculature with an incorrectly narrowed lumen of the bypass (see zoom-in). (b) displays a CR which accounts for the correct lumen of the bypass, but shows artifacts resulting from the trade-off between vessel visibility and the displayed size of the surrounding tissue (see zoom-ins). The vessel radius is offset by 150 pixels and the arc-length threshold equals to 300 pixels. (c) presents CSR showing the correct lumen of the bypass and accounting for proper overall vessel visibility. (d) demonstrates a top-right view of the vasculature using CSR together with MIDA as context visualization.

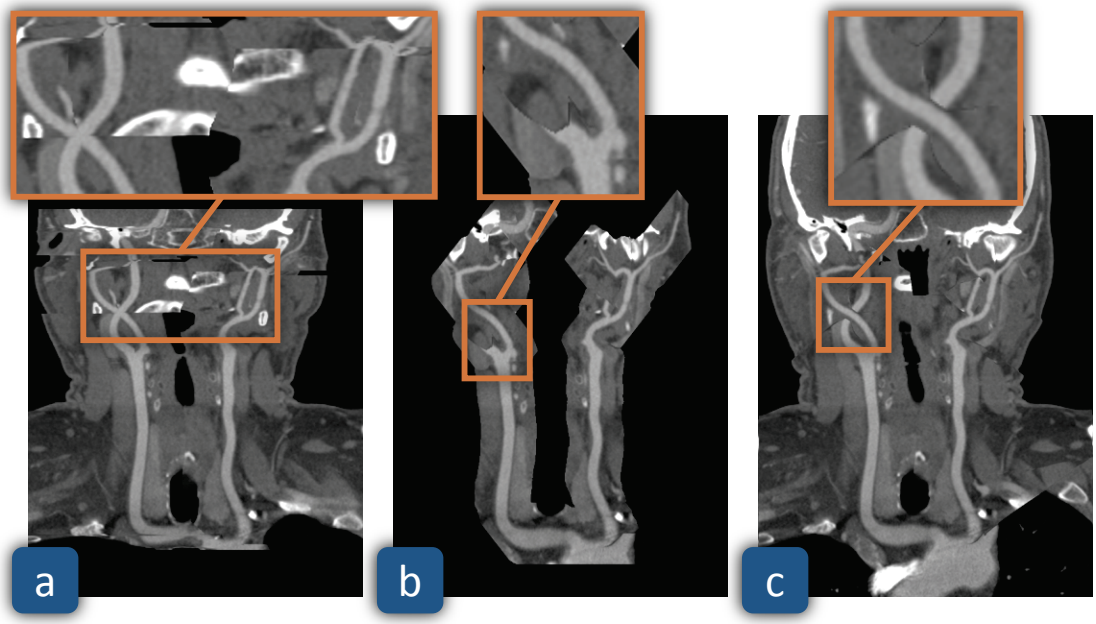


Figure 3.8: Cervical vessels of a human head CTA data set. (a) shows an mpCPR of the vasculature, which does not account for correct visibility (see zoom-in). (b) displays the lumina of the vessels using CR with a small vessel radius offset (50 pixels) and arc-length threshold (150 pixels) exhibiting a visibility artifact caused by the trade-off. The zoom-in shows an artifact coming from the large arc-length threshold, but when reducing it, visibility would further suffer. (c) presents CSR generating a smooth cut surface through the vasculature and surrounding tissue while preserving the visibility (see zoom-in).

Figure 3.6 presents a CTA data set of a human abdominal aorta at its first bifurcation displayed for several view directions. A DVR visualization is shown in Figure 3.6a together with the vessel tree rendered as wireframe model in red. Figures 3.6b-d give results of our method together with MIDA as context visualization. Visibility is correctly resolved in Figure 3.6b, as the left vessel branch is partially hidden due to its larger distance from the viewer. The cut surface is smooth even far from the vessel due to our LOD approach and exhibits intended jumps in depth when changing from the influence zone of one branch to another one. Figure 3.6c shows a view from bottom right, which depicts the vessel lumen and surrounding tissue. The reason for the smoothness is again the continuous transition between different LODs of the vessel tree. A view from the bottom is presented in Figure 3.6d. Here, a cutaway is generated to show the lumen of the aorta, which is located farther away from the viewer than its two branches.

Figure 3.7 shows a CTA data set of the vasculature of the human lower extremities with a cross-over bypass below the bifurcation of the abdominal aorta. We compare mpCPR (3.7a) with CR (3.7b) and CSR (3.7c). Although mpCPR provides a continuous

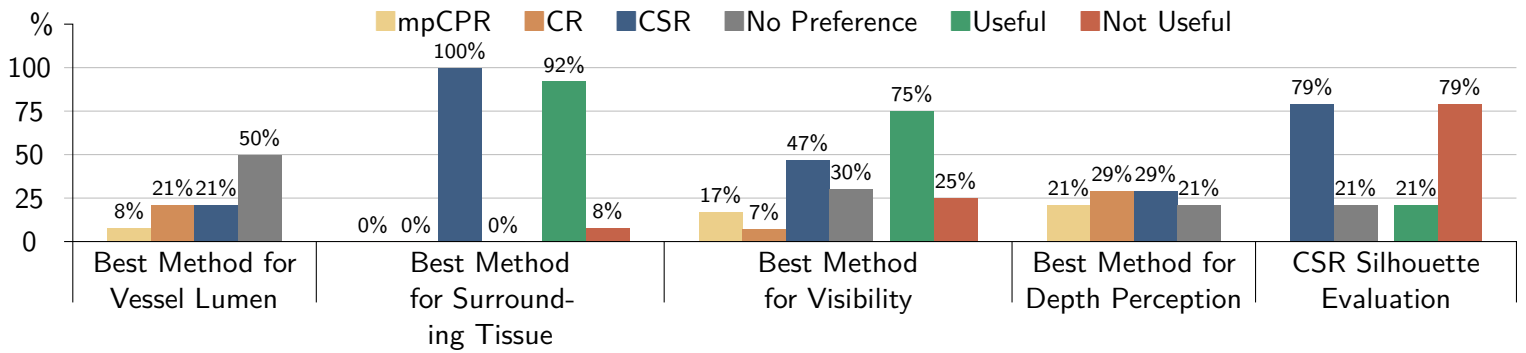


Figure 3.9: Evaluation of CSR in comparison to mpCPR and CR according to five criteria. The vertical axis shows the percentage (%) of the participants, where 100% equals to 12 domain experts. The preference of the participants was evaluated in the first three criteria, while in the last two, the number of correct answers has been counted. While no technique has clear advantages for inspecting the vessel lumen in a standard medical case, our technique compares favorably concerning the depiction of surrounding tissue and vessel visibility. The depth perception was roughly equal for all techniques and silhouettes provided no additional benefits.

cut surface through the vasculature and its surrounding tissue, it cannot visualize the lumen of the bypass adequately. The lumen is incorrectly narrowed, since the bypass runs horizontally, a situation that cannot be properly handled by mpCPR. This issue is addressed by CR, but a trade-off between the display of the surrounding tissue and the vessel visibility has to be made. This leads to artifacts, as pointed out in the zoom-ins, coming from the fact that the information of the left vessel overwrites information of the right one. CSR avoids these issues by creating the lumina entirely in 3D. Also the visibility of the denser vasculature in the lower legs is correctly preserved. Figure 3.7d shows a rotated view of the lumina of the vasculature using CSR. In order to enhance spatial perception, the cut surface is embedded into a context visualization.

A CTA data set of the human cervical vessels, including the carotid arteries, is shown in Figure 3.8. With mpCPR (3.8a), visibility is not preserved and the lumen of horizontally running vessels is not rendered correctly (see zoom-in). The surrounding tissue is, however, shown. With CR (3.8b), a trade-off between the display of the surrounding tissue and the visibility of the vessels has to be done. This is the reason why one vessel is cut off, as the vessel lies behind surrounding tissue (see zoom-in). Correcting the visibility would further limit the size of visible surrounding tissue. CSR (3.8c) preserves the visibility as well as provides a smooth cut through the surrounding tissue.

3.5 Evaluation

We consulted 12 radiologists from three different hospitals for an evaluation of CSR in comparison to mpCPR and CR with a questionnaire (see Appendix A). For a general assessment, the data set of Figure 3.6 was used, while the visibility was investigated on cervical vessels (see Figure 3.8). The questionnaire consisted of 21 questions from five criteria (see horizontal axis of Figure 3.9). Additionally, the usefulness of displaying the surrounding tissue, the visibility and the silhouettes were investigated, while the other two criteria are considered useful per se. The number of votes for the usefulness of the corresponding criterion is depicted as green bar, while the red bar shows votes doubting its usefulness.

There is no clear preference for a reformation technique for inspecting the *vessel lumen* in a standard medical case. mpCPR is rated lowest, but most participants see no clear difference as indicated by the large gray bar. This question should test if our method is equivalent to currently used clinical methods for standard cases.

All participants preferred CSR for investigating the *surrounding tissue*, and the domain experts regard this criterion as very useful (green bar). One radiologist specifically mentioned that the surroundings are beneficial for detecting other pathologies. This supports the LOD approach for smoothly extending the cut surfaces into the surrounding tissue.

For assessing the *visibility* of overlapping vessels, we use a cervical vessel data set (see Figure 3.8). The majority of participants rated CSR highest. The other categories in

order of popularity were no preference, mpCPR, and CR. Additionally, most domain experts considered a correct visibility as very useful (green bar). In fact, a smooth cut through the surrounding tissue while preserving the visibility of overlapping vessels was one of the motivations for our technique.

Regarding the *depth perception* of different vessels, CR and CSR are rated on the same level, mpCPR is rated worse. *Silhouettes* were not regarded as useful by most participants. We assume, however, that they could prove helpful after an accommodation period or for a complex vasculature, as specifically mentioned by one domain expert. A larger user study would be required to evaluate this effect as well as our context rendering.

Concerning the spatial orientation, domain experts pointed out that they use their anatomical knowledge to orient themselves and navigate within the 3D data sets and images. One radiologist specifically mentioned that visualizing the lumen of arbitrarily oriented vessels and having the possibility of unrestricted rotations, has great potential. In fact, this is part of the rationale for developing this method.

The feedback from domain experts indicates that CSR compares favorably to mpCPR and CR. Moreover, it enhances the visualization of surrounding tissue and the visibility of multiple overlapping vessels.

3.6 Discussion

In this chapter, we proposed with CSR a novel reformation technique that generates a cut surface through the lumen of vessels entirely in 3D. It does not require adjustment of any parameters by the user, while preserving correct visibility of vessels and their surrounding tissue. Moreover, arbitrary rotations are supported, giving the user complete freedom of inspecting the vessel lumina from every desired view direction. By relying on sample-based methodologies to resolve the visibility and to filter the density values of the imaging data, it offers interactive visual analysis of vessel pathologies.

We see several avenues for future work based on our method. As it allows the inspection of a complex vasculature in 3D, a given CSR view could be linked with a detailed view of a single vessel that contains aggregated information, such as based on curvicircular feature aggregation (Mistelbauer et al. 2013). The application to different curve-like anatomical structures such as nerves might be interesting. While we only evaluate our method's merits for vessel reformation, it is a general approach to generate view-dependent 2D surfaces along 1D curves that produces high-quality cuts through the surrounding 3D volume. The visualization of cut surfaces along curve-like features in scalar (Correa et al. 2011) and vector data (Jankun-Kelly et al. 2006; Kenwright and Haimes 1998) in various scientific and technical domains are possible application scenarios.

CHAPTER 4

Prefiltering on Polytopes

THIS chapter starts the exposition on prefiltered anti-aliasing. We present closed-form solutions for convolution filtering of linear functions defined on polytopes in n dimensions. Through the use of radially symmetric filter kernels, we ensure an isotropic response of the filter. Explicit formulas are provided for two and three dimensions and subsequently implemented in a rasterization (resp. voxelization) framework on graphics hardware. Furthermore, an evaluation in terms of anti-aliasing quality and runtime is presented. Note that this method requires input of the same dimensionality as the output, as we do not account for possible occlusions during a projection operation. We will remedy this limitation in Chapter 5, where we present an exact visibility algorithm.

4.1 Analytic Integration

The mathematical formulation of convolution prefiltering is as follows: Given a data function $\mathcal{I}$ defined on $\mathbb{R}^n$ and a filter function $\mathcal{F}$, we can obtain the filtered value $v(\boldsymbol{x})$ at sample location $\boldsymbol{x}$ by evaluating the convolution

$$v(\boldsymbol{x}) = \int_{\mathbb{R}^n} \mathcal{I}(\boldsymbol{y})\mathcal{F}(\boldsymbol{x}-\boldsymbol{y})\,d\boldsymbol{y}. \tag{4.1}$$

Examples for $\mathcal{I}$ are colors on 2D shapes or the density values inside objects.

In general it is not possible to derive a closed form solution for (4.1). With suitable conditions on the data term, its support, and the filter function, such a symbolic evaluation is achievable as shown by Duff (1989). His work covers analytic anti-aliasing in two dimensions utilizing separable polynomial filters and polynomial data functions defined on polygons. While these integrals can be solved analytically, the resulting formulas become unmanageable for higher dimensions. Our approach of using radially symmetric filter functions allows each filter order to be integrated separately and produces a single expression; hence, for m filter orders we have m summands *irrespective of the*

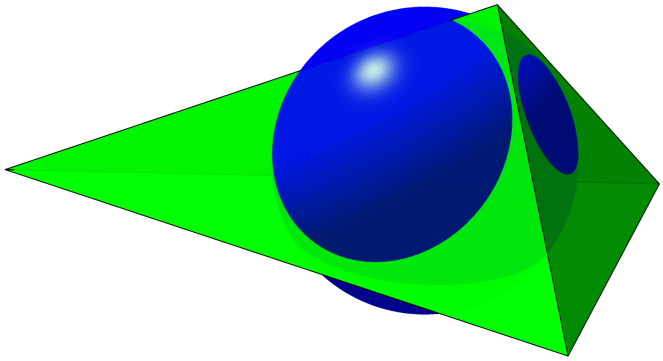


Figure 4.1: The intersection of a polytope with the filter support's sphere can generate complicated shapes, that have to be subdivided in order to obtain well behaved domains of integration.

dimension. A similar procedure for a separable filter would require the evaluation of all possible monomials, i.e. $x_1^{m_1} x_2^{m_2} \cdots x_n^{m_n}$, and would yield m^n summands for m filter orders and dimension n. Obviously, this becomes intractable for higher dimensions and one could try to obtain a single expression for all filter orders at once. We assume in this work that data functions are defined on polytopes—a situation that constitutes the most common use case for computer graphics applications—and consequently, the integration domains of the filter convolution is a polytope bounded by hyperplanes. For the integration along coordinate x_i, the boundary would be defined by an expression of the form $a_0 + \sum_{j=1}^{i-1} a_{i,j} x_j$. The result would thus depend on $\frac{1}{2}\left(n^2 + n + 1\right)$ variables $a_{i,j}$ for the integration boundaries alone. Together with the integration of all filter orders in *each* dimension and a polynomial data function this amounts to formulas of unmanageable sizes for three or more dimensions.

4.1.1 Setting

With spherically symmetric filters, our problem statement is formally as follows: Let $\mathcal{P}$ be a orientable non-self-intersecting polytope in $\mathbb{R}^n$, $\mathcal{I}_{\boldsymbol{a},c}(\boldsymbol{x}) = \boldsymbol{a} \cdot \boldsymbol{x} + c$ a linear function and $\mathcal{F}_R(\boldsymbol{x}) = \sum_{i=0}^{N} c_i \|\boldsymbol{x}\|^i \chi_{\|\boldsymbol{x}\| \leq R}$ a filter function with a cutoff radius R. For dimensions $n = 2, 3$ we will give a closed form solution for the convolution term

$$v(\boldsymbol{x}) = \int_{\mathcal{P}} \mathcal{I}_{\boldsymbol{a},c}(\boldsymbol{y}) \mathcal{F}_R(\boldsymbol{x} - \boldsymbol{y})\, d\boldsymbol{y} \tag{4.2}$$

$$= \int_{\mathcal{P} \cap \mathcal{S}_R} \mathcal{I}_{\boldsymbol{a},c}(\boldsymbol{y}) \mathcal{F}_\infty(\boldsymbol{x} - \boldsymbol{y})\, d\boldsymbol{y} \tag{4.3}$$

at the sample location $\boldsymbol{x}$. $\mathcal{S}_R$ denotes the filter function's support - a sphere with radius R. We present the case of two and three dimensions in detail below. The accompanying closed-form expressions and a generalization to higher dimensions is presented in Appendix C.

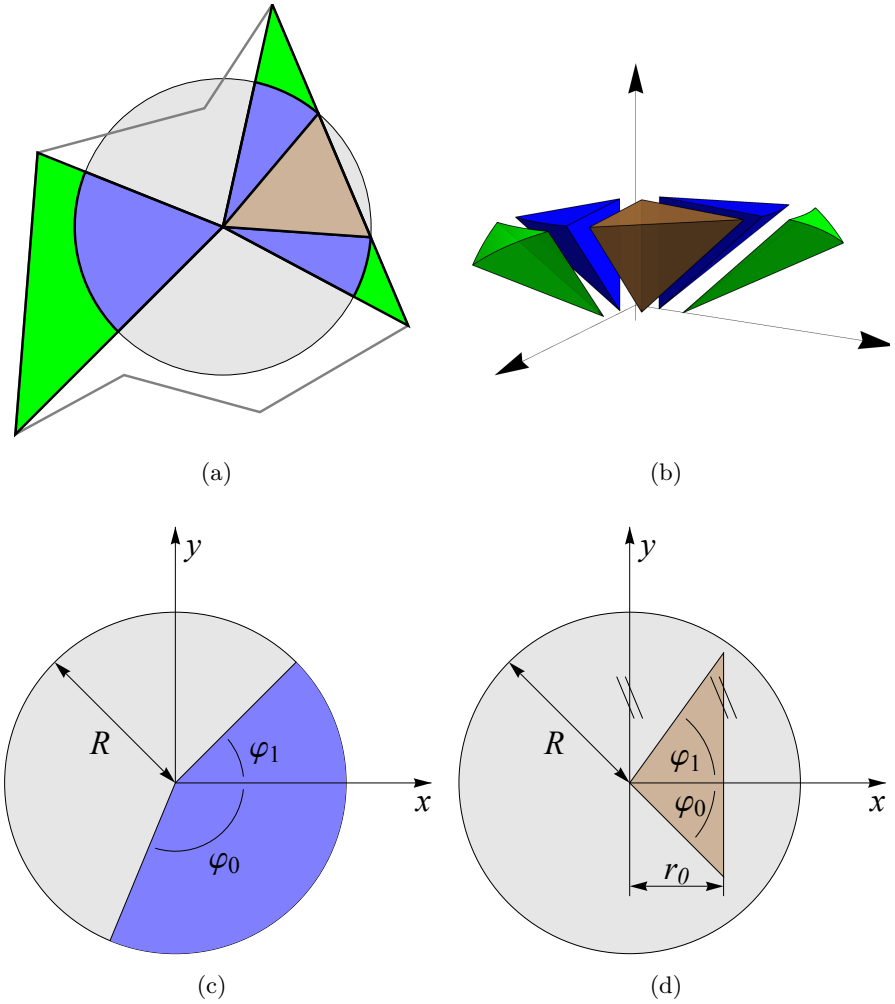


Figure 4.2: Polygon-circle intersection in two dimensions. Intersecting a polygon with the circular filter support gives three qualitatively different domains for subsequent integration (a). These domains can be immediately used for integration in the two-dimensional case. In three dimensions, these planar regions have to be extruded to the origin to yield the desired integration domains. The exploded view of such an extrusion of the right highlighted region of (a) is shown in (b). The marked variables for the sector (c) (resp. segment (d)) coincide with the 2D integration result in (C.1) (resp. in (C.2)).

The main difficulty in analytically computing this integral lies in its potentially complicated domain of integration, i.e. $\mathcal{P} \cap \mathcal{S}_R$. Especially in three or more dimensions the intersection of the filter's spherical support with a general polytope can produce a complicated shape on which the integrand has to be evaluated, as shown in Figure 4.1. Therefore, an important component of our solution is the partition of the intersection domain into a set of simple regions. Each of these regions falls into one of a small number

of geometric categories for which the integral (4.3) can be evaluated. We furthermore use the fact that by substitution the convolution can be rewritten as

$$v(\boldsymbol{x}) = \int_{\mathcal{P}} \mathcal{I}_{\boldsymbol{a},c}(\boldsymbol{y}) \mathcal{F}_R(\boldsymbol{x}-\boldsymbol{y})\, d\boldsymbol{y} = \int_{\hat{\mathcal{P}}} \mathcal{I}_{\hat{\boldsymbol{a}},\hat{c}}(\boldsymbol{y}) \mathcal{F}_R(\boldsymbol{y})\, d\boldsymbol{y} \tag{4.4}$$

with $\mathbf{R}$ being a rotation matrix, $\hat{\boldsymbol{a}} = \mathbf{R}^{-1}\boldsymbol{a}$ and $\hat{c} = c + \boldsymbol{a}\cdot\boldsymbol{x}$. We denote by $\hat{\mathcal{P}} = \mathbf{R}(\mathcal{P}-\boldsymbol{x})$ the shifted and rotated version of the polytope $\mathcal{P}$. This allows us to assume the filter to be centered at the origin and enables us to align a chosen face of the shape $\mathcal{P}$ with a given (hyper)plane by rotation around the origin. In comparison to the separable case, this gives us two main advantages. The filter is a function of just one coordinate, i.e., the radial direction, and the integration domains can be rotated to be 'as axis aligned as possible' thus yielding simple integration boundaries.

It should be noted that equation (4.4) also holds under anisotropic scaling of the space. Together with the invariance under rotation, we cover *elliptic* filters as well. Our framework additionally accommodates *piecewise polynomial* filter functions by evaluating the convolution for different filter radii, i.e., the convolution with two polynoms $a(\boldsymbol{x})$ and $b(\boldsymbol{x})$ which are defined on $[0, r_a]$ and $[r_a, r_b]$ can be treated by the sum of the convolutions with $a(\boldsymbol{x}) - b(\boldsymbol{x})$ on $[0, r_a]$ and $b(\boldsymbol{x})$ on $[0, r_b]$. Hence, for n piecewise polynomials the algorithm has to be executed n times.

4.1.2 Integration in Two Dimensions

We first outline the integration in two dimensions as the three-dimensional case builds upon it. Without loss of generality we assume the polygon boundary to be oriented counterclockwise and we determine the integral contributions of all polygon edges separately (see figure 4.2a). This leads to the correct result since the polytope was assumed to be self-intersection-free. Each edge—together with the filter center at the origin—spans a triangle, and thus it is possible to evaluate the integral simply by summing up the individual triangle contributions. Preserving the boundary orientation in the intersection procedure below leads to the correct signs in the final summation, similar to the work of Guenter and Tumblin (1996).

In this setting the intersection of the filter support $\mathcal{S}_R$ with such a triangle results in a simple line-circle intersection of the triangle edge that stems from the original polygon, and a centered circle with the cutoff radius R of $\mathcal{S}_R$. Depending on the edge geometry this leads to a varying set of *segments* and *sectors*. We denote with a *segment* a triangle that has one vertex at the origin and is fully contained in $\mathcal{S}_R$ (see Figure 4.2d). In contrast, a *sector* is a circular region of $\mathcal{S}_R$ (see Figure 4.2c). As can be seen in Figure 4.2a, boundary parts outside $\mathcal{S}_R$ result in sectors while the parts inside result in segments. Note that the same holds if the origin is not contained in the polygon as all contributions cancel correctly.

The contribution of a sector can be evaluated analytically as given in (C.1). Prior to integration we rotate a segment such that it lies in the positive half plane $x \geq 0$ and such

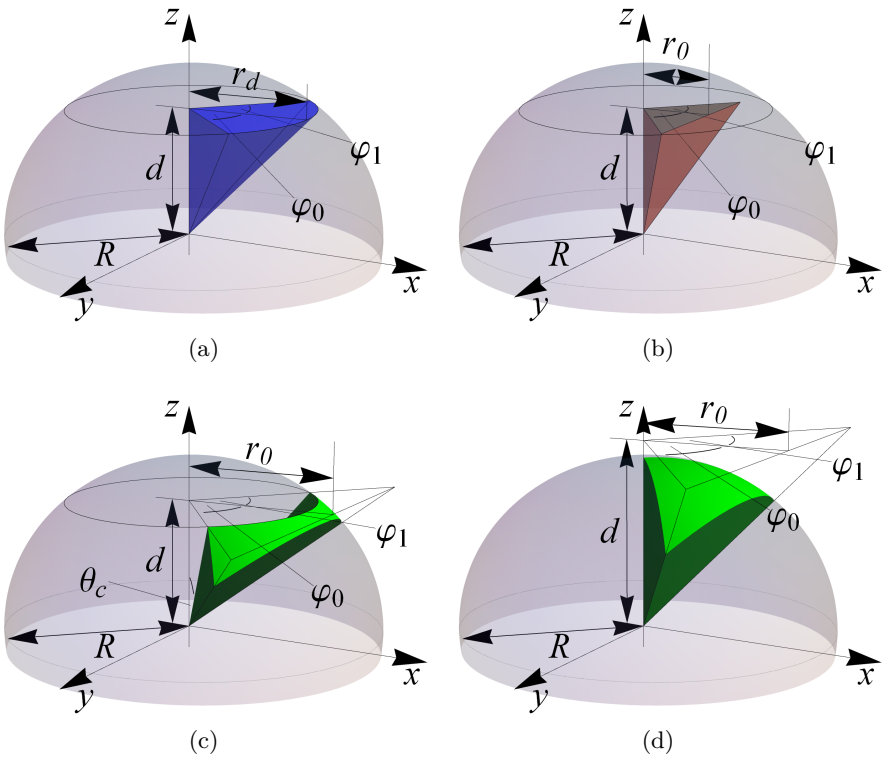


Figure 4.3: Integration domains in three dimensions. The three-dimensional domains of integration are obtained by extruding the two-dimensional intersection domains to the origin. The extrusion of a sector (see Figure 4.2c) gives the wedge of a cone (a) while a segment (see Figure 4.2d) yields the tetrahedron (b). The green appendage (see Figure 4.2a) has to be projected onto the filter support's sphere and gives a pyramid with a curved base (c). A special case of it is given in (d). Note that the color scheme is the same as in Figure 4.2 and that domains (a) and (c) always occur in pairs.

that the polygon boundary is parallel to the y-axis. This is shown in Figure 4.2d and the closed form solution is given in (C.2). Of course, the data function always has to be rotated together with the segment.

4.1.3 Integration in Three Dimensions

The three-dimensional case is a natural extension of the situation in two dimensions. Here, we assume a polyhedron with outward facing normals and all polygon faces oriented accordingly. Together with the filter center at the origin, each face spans a pyramid. As before, the integration over the whole polyhedron is the sum of the contributions of all individual pyramids. Each pyramid is again rotated around the origin such that it lies in the positive half space $z \geq 0$ and its base lies in the $z = d$ plane where d denotes the pyramid height. As stated in Equation (4.4), this does not alter the value

of the integration, as long as the data term is rotated correspondingly. Similarly, the orientation of the polyhedron faces are preserved up to the actual integration and the final summation.

Depending on the pyramid height d and the filter cutoff radius R, we have to consider two cases. If $d \leq R$, the pyramid's base polygon intersects the filter support $\mathcal{S}_R$, otherwise it does not. We treat the former case first as the latter one can be treated as a special subcase of it. The intersection of the pyramid's base polygon with $\mathcal{S}_R$ is done as in the two-dimensional case, with the difference that we use the cutoff radius r_d at height d which amounts to $\sqrt{R^2 - d^2}$. Apart from sectors and segments, a third kind of integration domain appears. In three dimensions, the area between the line segment that lies outside the circle and the resulting sector has to be treated as well and we refer to it as *appendage* (see the green areas in Figure 4.2a).

Since the base polygon lies at $z = d$, we have to incorporate the third dimension into our integration domain. A sector becomes the wedge of a cone, as shown in Figure 4.3a, and its value is given in Equation (C.3). Both the segment and the appendage are rotated around the z-axis such that their boundary edge opposite the vertex in $(0, 0, d)$ is parallel to the y-axis. The segment together with the origin spans a tetrahedron as can be seen in Figure 4.3b and its integration is given in Equation (C.5). The appendage produces a rather different shape (see Figure 4.3c). Projecting the area of the appendage onto the boundary sphere of $\mathcal{S}_R$ towards the origin results in an area that is confined from above by the circle with center $(0, 0, d)$ and radius r_d and from the sides by the great circles of the sphere with constant angle φ. The lower boundary is given by the projection of the relevant part of the polygon's edge onto the sphere. The final three-dimensional shape given by the appendage is the 'pyramid' with apex in the origin and a curved base section given by this projection. It integrates to the expression in Equation (C.4).

We are left with the aforementioned case where $d > R$ and the pyramid base does not intersect $\mathcal{S}_R$. Here we have no sectors nor segments but only an appendage (see Figure 4.3d). In this case we project it onto the filter support sphere $\mathcal{S}_R$ in the same way as done before and, by setting $\theta_C = 0$, we get the same result (see Equation (C.4)) as in the intersecting case.

4.2 Implementation

It is obvious that the evaluation of the convolution integral (4.3) can be done in parallel for each sampling location and each polytope, whereas the final summation can be done independently for each sampling location. This maps the problem very well to highly parallel hardware such as Graphics Processing Units (GPUs). We implemented the two-dimensional case with DirectX 10 and the three-dimensional case with Compute Unified Device Architecture (CUDA). The algorithm can be naturally divided into three stages: a setup stage, the intersection routine and a final integration. Due to the finite extent of the filter it is clear that if a sampling point $\boldsymbol{x}$ is placed outside the Minkowski sum of the current polytope and the filter support—as shown in Figure 4.4—the convolution

integral evaluates to zero. This fact localizes computations and is used in the setup stage to generate bounding regions for each input polytope to mask irrelevant output locations.

While the mathematical framework developed in section 4.1 works for general polygonal and polyhedral input, we implemented the most common case of input triangles and tetrahedra. In this case a linear interpolation of vertex values coincides with the linear data function.

The DirectX implementation for the two-dimensional case performs the setup stage in the geometry shader. Here, each incoming triangle is 'thickened' so that all pixels in the relevant region get rasterized (see Figure 4.4). The intersection and integration stages are then computed in the pixel shader. All operations are performed in a single render pass, making the technique easy to implement in existing rendering systems.

In the CUDA implementation for the three-dimensional case, the setup stage takes each incoming tetrahedron and displaces all four faces of it outwards by the filter support diameter. Similarly, the axis aligned bounding box of the tetrahedron is enlarged by the same amount. The intersection of these two figures tightly encloses the relevant region and is used as a mask in the following stages. The intersection stage takes as input a tetrahedron, a sampling location, and the spherical filter support. It generates the integration domain regions (sector, segments and appendages) as output. Every translation and rotation that is applied to the shape has to be applied similarly to the linear function defined on that shape, as stated in Equation (4.4). The generated domains are then processed in a separate integration stage which calculates the contributions to the final sum at each sampling location. Note that since the intersection of a tetrahedron and the filter support can result in a highly varying number (9-45) of integration domains, significant code path divergence can occur in the intersection stage. This could have significant performance impact, as threads with a large number of domain count could force threads with lower counts to idle. Consequently, we buffered the domain data and decoupled the intersection and integration stages in our CUDA implemention. Thus, each Single Instruction Multiple Data (SIMD) unit in the integration stage is assigned integration domains of the same type but from potentially different sample locations. For each integration domain type, a separate kernel is launched. We observed that the amount of computations needed for each stage effectively hides the memory latency of the GPU.

4.3 Results

We found that our analytic anti-aliasing method compares favorably against sampling-based approaches. Figure 4.5 shows a zone plane pattern consisting of 14400 colored triangles that form 80 rings with 4 degrees angular resolution. Anti-aliasing is performed with a Gauss filter with a radius of 2.3 pixels at a resolution of 400x400 pixels. The left image was computed using our prefiltering method. It took 7.4 ms to compute on a GeForce GTX 580 graphics card using a single DirectX shader pass. The right image was computed in 7.9 ms employing optimized jittered supersampling to obtain 576 samples.

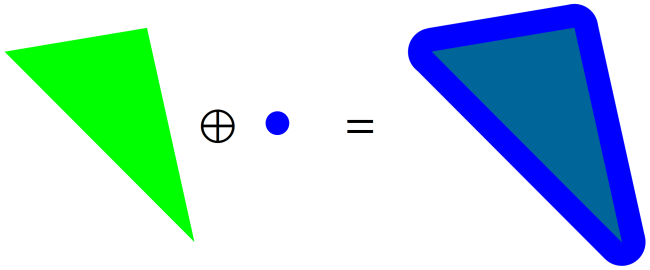


Figure 4.4: Taking a triangle (left) and a radial filter support (center) as input, their Minkowski sum is given as the union of all filter locations that intersect the triangle (right).

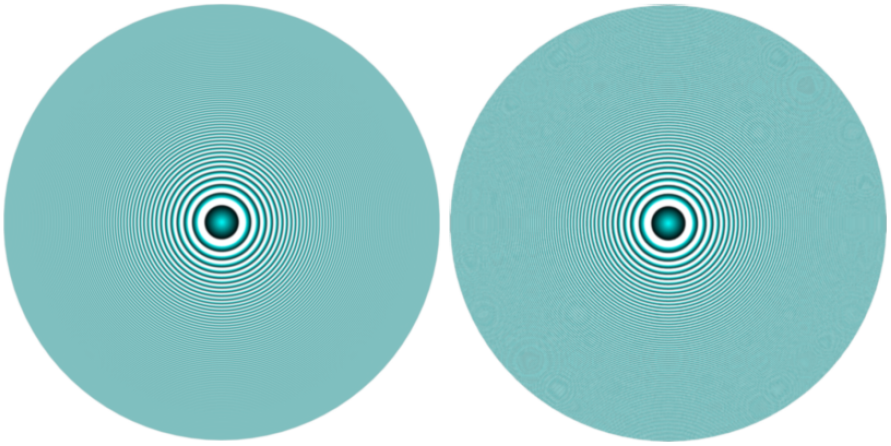
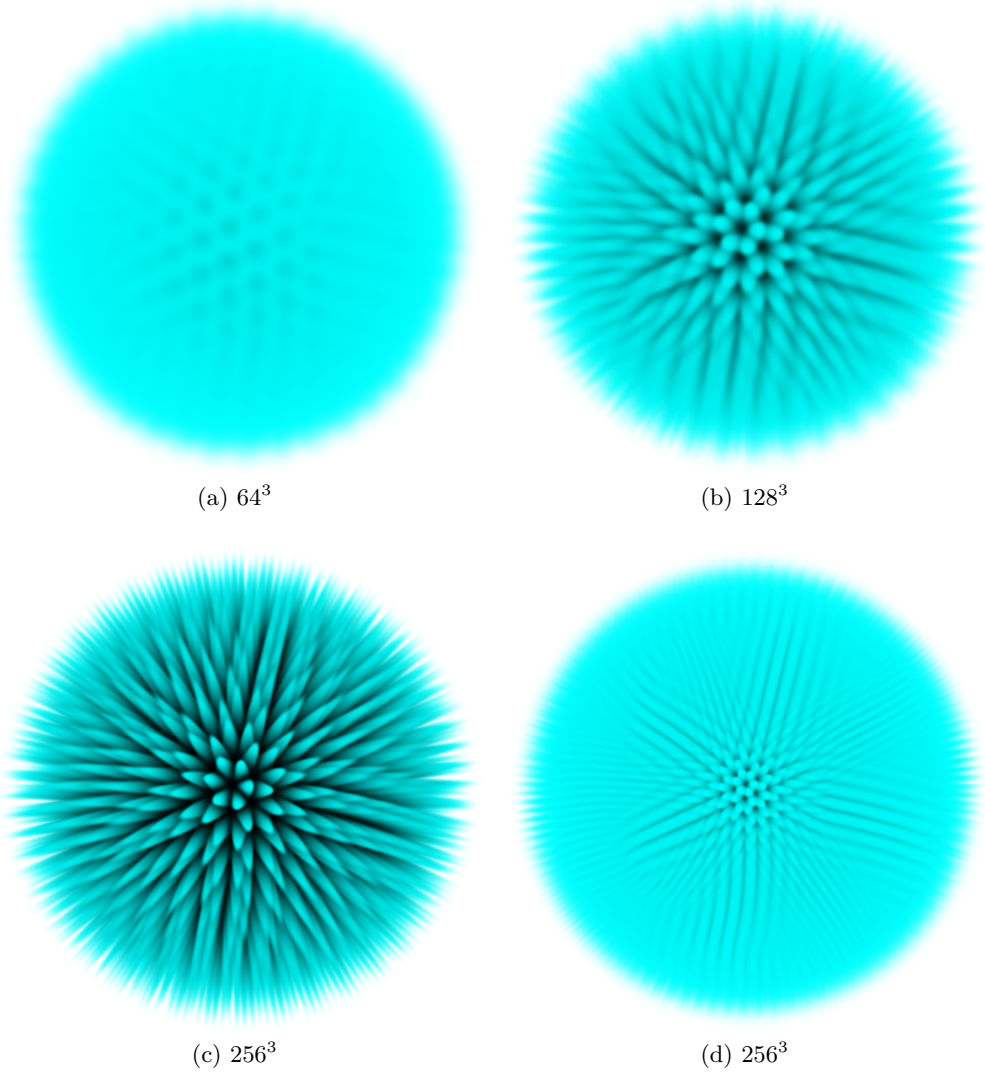


Figure 4.5: Prefiltering and supersampling comparison. In an equal time comparison, our prefiltering method (left) outperforms jittered supersampling (right) on a scene with high anti-aliasing requirements.

A Halton sequence efficiently places samples and each sample contributes to all pixels in its filter range by using the common GPU rasterizer. While this causes correlations between the pixel samples, a completely stochastic sampling approach using General-Purpose Computing on Graphics Processing Units (GPGPU) would induce additional bandwidth and computations and might undo the benefits from sample decorrelation. For this scene with very high anti-aliasing requirements, there still remain aliasing artifacts when using supersampling. We want to note that for typical scenes in real-time applications, anti-aliasing requirements are considerably lower and the slow convergence rate of supersampling-based methods is less problematic.

An overview of executing times is given in Table 4.1. For the case of anti-aliasing voxelization in three dimensions, we state the runtime for different input complexities and grid resolutions. The intersection stage constitutes the bottle-neck, mainly due to



(a) 64^3 (b) 128^3

(c) 256^3 (d) 256^3

Figure 4.6: Direct volume renderings of a colored sea urchin with different complexities at various spatial resolutions (given as captions) using a Gaussian filter with 2.3 voxels filter radius. The tetrahedron count for (d) is 12652 and 2470 for (a)-(c). The base of the spikes is colored black while the tip vertex is in cyan. The color value is linearly interpolated inside the spike tetrahedra. Note that for decreasing spatial resolution (a)-(c) or higher frequencies in the source data (d) the output shows no aliasing effect.

Figure 4.7: Direct volume renderings of an extruded zone plate pattern. The input data consists of nearly two million tetrahedra and was analytically filtered with an area filter (a)-(d) and a Gaussian filter (e)-(h) to a Cartesian grid with dimensions 128^3. Each filter was evaluated with support radii 1, 2, 2.5 and 3 (left to right). For too narrow radii both filters show severe aliasing (a)&(e). While the most glaring artifacts disappear with increasing radii, the area filter still exhibits defects at low frequencies which are not present when using the Gaussian filter. The rendering technique uses linear interpolation of the source data and a linear transfer function.

	Grid resolution								
	64^3			128^3			256^3		
Tetrahedra	$\cap$	$\int$	$\sum$	$\cap$	$\int$	$\sum$	$\cap$	$\int$	$\sum$
19.0k	2.0	0.5	2.5	11.7	2.0	13.8	78.6	12.0	90.5
190.0k	4.2	1.1	5.2	18.9	4.1	22.9	111.7	19.0	130.6
1.9M	12.1	3.1	15.3	38.1	9.8	47.8	179.9	37.0	216.8

Table 4.1: Runtime in seconds to sample a cube—composed of a varying number of tetrahedra—to Cartesian grids of increasing resolutions. The timings for the intersection stage ($\cap$), the integration stage ($\int$), and their sum ($\sum$) is given. A Gaussian filter function with five even polynomial orders and a radius of 2 grid cell lengths was used.

the inevitable code path divergence caused by different geometrical configurations. Since the number of intersections is not known in beforehand, a initial load balancing stage cannot be used to improve this situation. However, complex tetrahedral models can be exactly filtered and rasterized into Cartesian grids within few seconds up to few minutes.

Figure 4.7 shows a 3D rasterization of an extruded zone plate using area and Gaussian anti-aliasing. It is clearly visible that the more complex Gaussian filter is able to remove aliasing at a smaller filter radius while also preserving larger features much better. This emphasizes the need for sophisticated filters that were not possible with previous approaches.

Finally, Figure 4.6 shows a sea urchin sampled at different spatial resolutions. The direct volume rendering shows how our analytic filtering smoothly filters the solid volume at different spatial resolutions.

4.4 Discussion

This paper we presented an analytic formulation for prefiltered sampling of polytopes with a linear function defined on them. This includes the practically relevant cases of triangles and tetrahedra with linearly interpolated vertex values. Together with a spherically symmetric polynomial filter, this allows the closed-form evaluation of the convolution integrals. We presented an implementation on graphics hardware and an evaluation in terms of runtime and anti-aliasing quality.

As extensions to this work. the linear volumetric approach might be used to render time-varying data, like for example the helicopter blades in the work of Dachille and Kaufman (2000). Elliptical filtering is made possible straightforwardly by scaling the polyhedra in the opposite direction to render effects like motion blur and support non-square pixels.

One general observation is that the filter convolution is evaluated only once for each pair of input primitive and sampling location. In certain configurations, this approach utilizes

less memory bandwidth than sampling-based approaches that provide similar quality. Although the arithmetic workload is much higher due to the closed-form solutions, it can be observed that the actual computing resources of most hardware designs are increasing considerable faster than memory bandwidth. In this sense, the relevance of prefiltering is growing as long as this development continues.

This approach converts n-dimensional input signal into raster representations of the same dimension, e.g., 2D rasterization or 3D voxelization. If a dimensionality reduction occurs—as, for example, for the highly relevant case of 3D-to-2D rasterization—possible occlusions have to be accounted for. We provide a technique to achieve this in a computationally efficient manner in the next chapter.

CHAPTER 5

Exact Parallel Visibility

In the previous Chapter 4, we introduced an analytic prefiltering method for anti-aliasing. In particular, rasterization of polygons to two-dimensional raster images and the voxelization of polyhedra to three-dimensional grids was presented. The arguably most common use case, which is 3D-to-2D rasterization, was not covered, however, as the projection operation of a three-dimensional scene onto a two-dimensional image plane is missing from this technique. In this chapter, the missing piece will be provided in the form of an exact hidden-surface algorithm. By removing all occluded parts from the input scene, a two-dimensional vector-format representation of the scene from a given viewpoint is generated, which can be directly used as input for the filtering procedure of Chapter 4. Together, these two methods provide a fully prefiltered equivalent to common rasterization, and thus a ground-truth solution to visibility and shading anti-aliasing.

5.1 Analytic Visibility

The task of resolving the exact visibility of polygonal scenes has a long history in computer graphics dating back to the advent of vector displays (Sutherland 1964). Many works throughout the years presented sequential algorithms to solve this problem and later parallelization assumed a small number of parallel processors. In this work, we explicitly target massively parallel hardware architectures and present novel methods to exploit their computational capabilities for the task of hidden-surface elimination.

The adaptation of a visibility algorithm to massive Single Instruction Multiple Data (SIMD) architectures has many objectives. Ideally, it should allow the workload to be split into a predictable and large number of small and independent subtasks. Furthermore, it should accommodate simple data structures that can be accessed in coalesced parallel fashion. Additionally, the actual computations should rely on basic data types such as integral or floating point values. These restrictions rule out methods that internally use arbitrary polygons (Weiler and Atherton 1977) or generate complex graphs (Dévai 2011),

(a) View plane projection (b) 2D view (c) Edge intersections (d) Hidden-line elimination (e) Boundary completion

Figure 5.1: Overview of our analytic visibility method. The scene triangles are projected onto the view plane (a) as required by the *edge intersection* phase (see Section 5.1.2), which operates mostly in 2D (b). It determines for each edge the intersections with all other triangles (c). The intersection data is used in the second phase to determine the *visible line segments* (see Section 5.1.3). A *hidden-line elimination* algorithm gives the visible segments of each edge (d) and a final *boundary completion* (e) completes all visible line segments. The final set of line segments constitutes all boundaries of the visible regions of the scene triangles.

as well as arbitrary precision arithmetics. Furthermore, acceleration data structures are needed to avoid the $O(n^2)$ complexity of intersecting all triangles with each other. Visibility algorithms generally contain a sorting routine (Sutherland et al. 1973), which has to be mapped to efficient methods on the Graphics Processing Unit (GPU).

5.1.1 Overview

Our visibility algorithm takes the normalized device coordinates of all 3D triangles as input, and outputs a list of all two-dimensional line segments, which constitute the borders of all visible polygonal regions of the scene when projected onto the view plane. It is assumed that the triangles are *consistently oriented* (to allow backface culling) and *non-intersecting*. Note that *cyclic overlap* of triangles is permitted. The result gives a complete description of the boundaries of all visible regions and matches the input requirements of analytic anti-aliasing approaches (Auzinger et al. 2012; Manson and Schaefer 2011).

Our method performs the visibility computation independently for each edge, which yields an embarrassingly parallel workload. Building on a method of Dévai (2011), we present a new edge-triangle intersection method that allows replacing the theoretical 'black hole' treatment of Dévai with a novel, implementation-friendly boundary completion stage. Both algorithms and the additional hidden-line-elimination stage are presented in the next sections.

5.1.2 Edge Intersections

To determine the visible parts of the scene edges, we first project them onto the view plane and compute the intersection in this space. Since projected triangles are convex sets, we can assume that their intersection with a projected edge e are connected subsets of e, i.e., the occlusion of a projected edge e by a projected triangle is a line segment contained in e. Since line segments can be stored by their respective endpoints, we have a fixed data size for the result of each edge-triangle-intersection. This enables efficient parallel read and write operations as offsets into a global buffer can be computed easily.

A formal description of our intersection algorithm is given in Algorithm 1 and an accompanying example in Figure 5.2. The core routines Intersect and IsBehind merit further discussion. The intersection routine Intersect reduces to a geometrical computation in 2D, as both its input parameters, edge e and triangle t, are projected onto the view plane. Note that we actually compute the intersection between the infinite line that contains e and the triangles. As such, it is possible that line segments that are *not* fully contained in e are reported. AreOverlapping is a conservative test to discard triangles that do not intersect e. Apart from point degeneracies, we utilize the fact that each intersections is represented by a line segment; Intersect outputs the segment's end points in 3D, where the depth coordinate is chosen such that the point lies on the plane of t. It should be noted that we discard single-point intersections in this phase, as zero-length line segments do not contribute to the final image. Additionally,

Algorithm 1 Edge intersection phase

Input: The set T of all projected scene triangles and the set E of all their edges.

Output: A set O with the intersection data, where $O(e)$ gives the intersection data for edge e. Each entry of $O(e)$ is the 3-tuple $(p, flag, type)$ consisting of an intersection point p, a flag that indicates if p is *starting* or *ending* a line segment, and a *type* describing the relative depth relation between e and p (i.e., one of *occluding*, *occluded*, or *self*).

```
1:  procedure EdgeIntersections(E, T)
2:      O ← {}
3:      for all edge e ∈ E do in parallel
4:          O(e) ← {}
5:          for all triangle t ∈ T, e ∉ t do in parallel
6:              if AreOverlapping(e, t) then
7:                  (p0, p1) ← Intersect(e, t)
8:                  if IsBehind(e, p0, p1) then
9:                      AppendTo(O(e), (p0, starting, occluded))
10:                     AppendTo(O(e), (p1, ending, occluded))
11:                 else
12:                     AppendTo(O(e), (p0, starting, occluding))
13:                     AppendTo(O(e), (p1, ending, occluding))
14:                 end if
15:             end if
16:         end for
17:         (p0, p1) ← EndPoints(e)
18:         AppendTo(O(e), (p0, starting, self))
19:         AppendTo(O(e), (p1, ending, self))
20:     end for
21:     return O
22: end procedure
```

each endpoint is marked as either *starting* or *ending* the occlusion. Since we will map the inner loop to a single SIMD unit, the AppendTo functionality can be efficiently implemented with intra-SIMD prefix sum operations to compute the offsets into the output buffer. Furthermore, such write operation will be coalesced.

For hidden-line elimination, it is sufficient to only consider the occlusions of an edge e (Appel 1967). Our goal, however, is full hidden-surface removal and—for reasons explained below—we also have to take those triangles into account that are occluded by e. The IsBehind routine of Algorithm 1 compares the relative depth of edge e and a line segment given by p_0 and p_1. Since we assume non-intersecting triangles as input, their depth ordering is well-defined and we store this information in *type* by denoting a line segment either as *occluded* or as *occluding*.

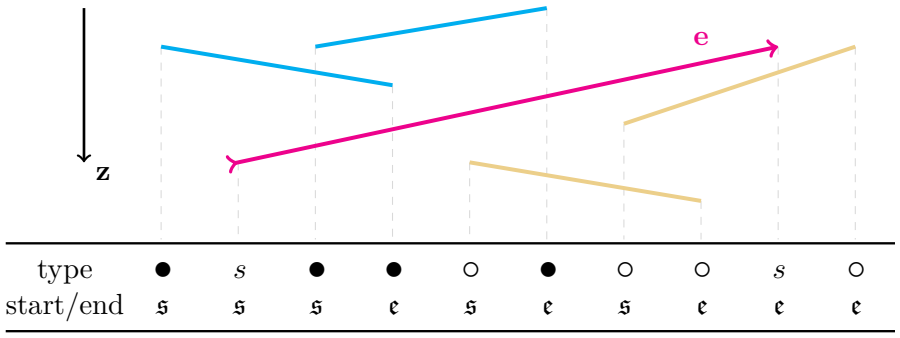


Figure 5.2: Example result of the EdgeIntersections procedure (see Algorithm 1). The output of a single iteration of the inner loop is shown for edge **e** (magenta). The plane that contains **e** and extends into z direction intersects two triangles in front of **e** (blue) and two triangles behind **e** (yellow). All four triangles overlap **e**, i.e., AreOverlapping evaluates to true. Thus, the intersection points are the endpoints of the associated line segments. Each point is associated with a type, denoting if **e** is occluded by its triangle (●), if it is an endpoint of **e** (s), or if its triangle is occluded by **e** (○). Additionally, a flag stores if the intersection points starts ($\mathfrak{s}$) or ends ($\mathfrak{e}$) an occlusion.

Great care has to be taken to ensure the robustness of such geometrical calculations on the employed hardware. Fixed precision or floating point arithmetics lead to round-off errors and can cause erroneous results in binary geometric decisions, such as the question if a given point lies on a given line. *Exact geometric computation* is the most commonly used technique to solve this problem. But since it relies on a combinatorial analysis of the geometric calculation at hand and on the use of arbitrary precision numbers, it is not suited for practical implementation on graphics hardware. We chose an ϵ-prefiltering approach by Frankel et al. (2004). In our implementation, this allows robust decisions even when using single precision 32-bit floating-point arithmetics. Only the degenerate case of parallel triangle edges requires the use of a double precision 64-bit floating point format. This enables us to reliably determine the correct intersections for geometrically complex models (see Figure 5.5).

5.1.3 Visible Line Segments

Having obtained the intersections for each scene edge, we employ two procedures to determine all visible line segments necessary for the final integration stage. We present this as two separate methods. The first is a *hidden-line-elimination* algorithm (see Figure 5.1d) while the second method completes the boundaries of the visible regions of each triangle and thus provides full *hidden-surface elimination* (see Figure 5.1e).

Algorithm 2 Determine the visible parts of all scene edges

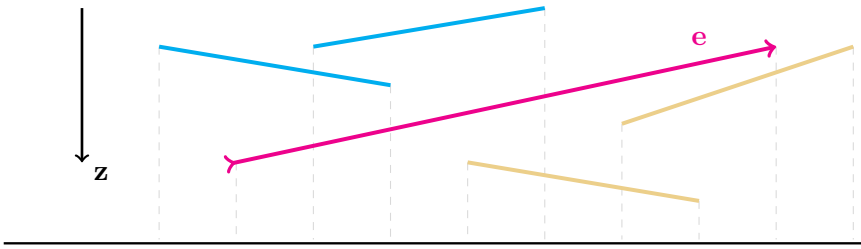
Input: The set E of all projected edges of all scene triangles. The *sorted* output O of EdgeIntersections (see algorithm 1).

Output: The visible line segments of all edges as reported by ReportSegment.

```
1:  procedure HiddenLineElimination(E)
2:      for all e ∈ E do in parallel
3:          I ← O(e)
4:          for n ← 1 to |I| do in parallel
5:              (∼, ∼, type) ← I(n)
6:              if type = occluding then RemoveIndex(I, n) end if
7:          end for
8:          Barrier
9:          for n ← 1 to |I| do in parallel
10:             (∼, flag, type) ← I(n)
11:             if flag = starting then V(n) ← 1 else V(n) ← −1 end if
12:             if type = self then V(n) ← −V(n) end if
13:         end for
14:         Barrier
15:         S ← ParallelInclusiveScan(V)
16:         Barrier
17:         for n ← 1 to |I| do in parallel
18:             v ← V(n), s ← S(n)
19:             if (v = 1 and s > 0) or (v = −1 and s > −1) then
20:                 RemoveIndex(I, n)
21:             end if
22:         end for
23:         Barrier
24:         for n ← 1 to n ≤ |I| step n ← n + 2 do in parallel
25:             (p0, ∼, ∼) ← I(n)
26:             (p1, ∼, ∼) ← I(n + 1)
27:             ReportSegment(p0, p1)
28:         end for
29:     end for
30: end procedure
```

Hidden-Line Elimination

Hidden-line elimination is a standard technique in line rendering and was first introduced by Appel (1967). Our method adapts a recent result of Dévai (2011) on the optimal runtime of parallel hidden-surface-elimination algorithms for our method. We use a scan-based approach which walks along the line of a given edge e and determines for each intersection point if it is an endpoint of a visible segment of e. This requires an



type	●	s	●	●	○	●	○	○	s	○
start/end	𝔰	𝔰	𝔰	𝔢	𝔰	𝔢	𝔰	𝔢	𝔢	𝔢
initial	1	-1	1	-1	✗	-1	✗	✗	1	✗
scanned	1	0	1	0	✗	-1	✗	✗	0	✗
output	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗

Figure 5.3: Example result of the HiddenLineElimination procedure (see Algorithm 2). The output and intermediate variables of a single iteration of the inner loop is shown for edge **e**. Note that the same scene as for Figure 5.2 is used. With the information obtained from the edge intersection phase (first two rows), an intermediate buffer V is initialized (initial) and—with the result S of prefix sum operation (scanned)—the intersection points marked with ✓ are reported as endpoints of visible parts of **e** (output). Entries removed by RemoveIndex are marked with ✗.

ordering of the intersections along e, which is achieved by an *intermediate sorting step*.

A formal description of hidden-line elimination assuming a sorted input is given by Algorithm 2 with an accompanying illustration given by Figure 5.3. For each edge in parallel, we first remove all intersections that originate from triangles that are occluded by e (see Lines 4-7). Depending on the type of each intersection point, we assign a value (0 or ± 1) to a list V (see Lines 9-13). Intuitively, V can be understood as the change in the number of occluders at each intersection, i.e., $+1$ if an occluder starts and -1 if one ends. The following ParallelInclusiveScan utilizes prefix sum computations to create a list S that holds a measure on how many triangles occlude e at a given intersection. We remove all occluded edge segments (see Line 20) and report the remaining ones as endpoints of visible edge segments. The criterion for removal (see Line 19) reflects the fact that for a given intersection index n, a visible edge segment starts when the last occluder ends (i.e., $V(n) = -1$ and $S(n) = -1$). The end of such a segment is given by the start of new occluder (i.e., $V(n) = 1$ and $S(n) = 0$). Note that the initial values at the endpoints of the edge e itself act as infinite occluders before and after e. For general architectures, memory barriers are necessary to prevent read after write data hazards. In our case, the implicit synchronization of the SIMD units already solves this issue without further effort and the Barrier instructions can be committed.

Algorithm 3 Determine the missing boundaries of the visible regions

Input: The set E of all projected edges of all scene triangles. The *sorted* output O of EdgeIntersections (see algorithm 1).

Output: The missing line segments of all edges as reported by ReportSegment.

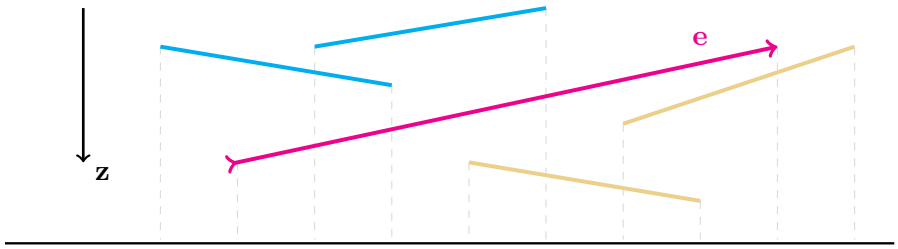
```
1:  procedure BoundaryCompletion(E)
2:      for all e ∈ E do in parallel
3:          I ← Data(e)
4:          for n ← 1 to |I| do in parallel
5:              Vis(n) ← IntersectionIsVisible(I, n)
6:          end for
7:          Barrier
8:          for n ← 1 to |I| do
9:              (∼, flag, type) ← I(n)
10:             if (type ≠ occluding or flag ≠ starting) then continue end if
11:             ID ← GetTriangleID(T, e, n)
12:             (n_start, n_end) ← GetIndices(I, ID)
13:             for k ← 1 to |I| do in parallel
14:                 if k = n_start or k = n_end then continue end if
15:                 (∼, ∼, type_k) ← I(k)
16:                 if type_k = occluding then
17:                     ID_k ← GetID(T, e, k)
18:                     F(k) ← IsInFront(ID_k, ID)
19:                 end if
20:             end for
21:             Barrier
22:             for k ← 1 to |I| do in parallel
23:                 (∼, flag_k, type_k) ← I(k)
24:                 V(k) ← Init(k, n_start, n_end, flag_k, type_k, Vis(k), F(k))
25:             end for
26:             Barrier
27:             S ← ParallelInclusiveScan(V, +1)
28:             Barrier
29:             I_n ← I(n)
30:             for k ← 1 to |I| do in parallel
31:                 v ← V(k), s ← S(k)
32:                 if (v = 1 and s > 0) or (v = −1 and s > −1) then
33:                     RemoveIndex(I_n, k)
34:                 end if
35:             end for
36:             Barrier
```

```
37:             for k ← 1 to k ≤ |I| step k ← k + 2 do in parallel
38:                 (p_0, ∼, ∼) ← I_n(k)
39:                 (p_1, ∼, ∼) ← I_n(k + 1)
40:                 ReportSegment(p_0, p_1)
41:             end for
42:         end for
43:     end for
44: end procedure

45: procedure Init(n, n_start, n_end, flag, type, visible, inFront)
46:     if n = n_start then return −1 end if
47:     if n = n_end then return 1 end if
48:     if type = occluding and inFront = true then
49:         if flag = ending then return −1 else return 1 end if
50:     end if
51:     if visible = true then
52:         if (type = self and flag = starting) or (type = occluded and flag =
    ending) then
53:             return −1
54:         end if
55:         if (type = self and flag = ending) or (type = occluded and flag =
    starting) then
56:             return 1
57:         end if
58:     end if
59:     return 0
60: end procedure
```

Hidden-Surface Elimination

To complete the boundary of each visible region, we propose an extension to the parallel hidden-line elimination of the previous section. In conjunction, they provide full hidden-surface elimination. Intuitively, we want to determine which triangles lie 'on the other side' of the visible edge segments. An example is illustrated in Figure 5.1, in which Figure 5.1e shows the missing line segments that complete the visible edge segments (see Figure 5.1d) to yield the complete boundaries of all visible regions (see Figure 5.1b). This is a harder problem than hidden-line elimination, as we are interested in the triangles that are right behind a given edge e (as opposed to just knowing that e lies behind one or more triangles). Formally described by Algorithm 3, we execute hidden-line elimination for every line segment o that is occluded by e (see Lines 8-42). The initialization of V is changed in such a way that the visible parts of e act as anti-occlusions, i.e., all segments o are occluded unless they lie behind a visible part of e (see Figure 5.4).



type	●	*s*	●	●	○	●	○	○	*s*	○
start/end	𝔰	𝔰	𝔰	𝔢	𝔰	𝔢	𝔰	𝔢	𝔢	𝔢
visible	F	F	F	F		T			T	
infront							T			T
initial	0	0	0	0	-1	-1	1	1	1	-1
scanned	1	1	1	1	0	-1	0	1	2	1
output	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗

Figure 5.4: Example result of the BoundaryCompletion procedure (see Algorithm 3) for the scene in figure 5.2. The output and intermediate variables of a single iteration of the inner loop is shown for edge **e** with an iteration step n such that the occluded line segment **o** is given by the intersection points with indices n_{start} and n_{end}. Note that the same scene as for Figure 5.2 is used. With the information obtained from the edge intersection phase (first two rows), an intermediate visibility buffer Vis stores the visibility of all intersection points of **e** (visible) whereas F denotes which occluded line segments are in front of **o** (infront). As in the HiddenLineElimination procedure (see Algorithm 2), an intermediate buffer is initialized (initial)—this time with an explicit procedure Init—and a 1-initialized prefix sum computed (scanned). Finally, the line segments of **o** just behind **e** are reported (output) and marked with ✓. Entries removed by RemoveIndex are marked with ✗.

In the initialization phase we also have to resolve the relative depth layering of the occluded line segments. Our algorithm achieves this by pairwise comparison of the occluded line segments. This is a standard geometric computation and is denoted by the procedure IsInFront in Line 18. As already mentioned, we report line segments that are directly behind edge e. This requires the knowledge of all visible line segments of e, which is exactly the result of HiddenLineElimination. We abbreviate it with the call of IntersectionIsVisible in Line 5.

5.2 Implementation

Our analytic visibility method targets massively parallel SIMD architectures with current GPUs as a prime example. We make use of their large amount of moderately sized SIMD units by implementing a software rendering pipeline using the Compute Unified Device

Architecture (CUDA) Application Programming Interface (API). We give a short review of this environment and continue with a detailed explanation of our design choices.

5.2.1 Hardware

In the following, we use NVidia's nomenclature and refer to the SIMD units as *warps* and assume their size to be 32 threads. They are grouped into *thread blocks* which enables the use of a processor's fast on-chip memory as *shared memory* by all the block's threads. The much larger and considerably slower global memory allows data transfer and synchronization across thread blocks, where the latter is enabled by atomic memory accesses. The amount of registers and shared memory is limited and the excessive use of one resource can decrease the amount of warps that can run in parallel, leading to device underutilization. While our design can also be implemented on other parallel hardware, such as multi-core Central Processing Units (CPUs), our algorithm benefits from the huge amount of threads that are kept active by a GPU, thus hiding memory latency.

5.2.2 Design Considerations

Our algorithm shows a distinct two-level parallelism in all its core procedures (see Algorithms 1-3). The outer loop iterates over all edges in parallel. Since the number of intersections per edge varies greatly, we cannot employ a SIMD model at this level without incurring severe code path divergence. Therefore, we assign edges to separate SIMD units and parallelize the inner loops across the threads of the respective units.

In the edge intersection phase (see Section 5.1.2) we assign each edge to a warp and fetch one triangle per thread until the pool of relevant triangles is depleted. Both the assignment of edges to warps and the computations of the offset into the output array are done with atomic memory functions on global counters. In our tests the large number of active warps efficiently hide the memory latency associated with the triangle fetches.

The sorting of intersections along the edges is executed for all edges in parallel by using a key-value radix sort. The key of each intersection holds a reference to its edge and a parameter that increases along the edge. Since the number of intersections per edge is known, the sorted intersection can be efficiently retrieved by the subsequent stages by offsetting.

The hidden-line elimination and boundary completion are executed similar to the edge intersection. The edges are assigned to warps and the inner loops parallelized across their threads.

5.2.3 Analytic Visibility Pipeline

In this section, we provide an overview of the actual pipeline we implemented and essential details on the adaptation to the CUDA framework as well as vital optimizations. Our algorithm to intersect edges with triangles has an outer loop over all edges and an inner loop over all triangles. A quadratic complexity in the number of triangles will become

prohibitively costly for large scenes and thus we assign the scene triangles to subdivisions of the view plane—in our case quadratic *bins*. This preserves spatial coherence in the memory accesses and enables load balancing by prioritizing bins with a high number of assigned triangles. We generate a list of overlapped bins per triangle and use radix sort (Merrill and Grimshaw 2011) to obtain a list of triangles per bin (similar to the work on voxelization by Pantaleoni (2011a)). Our use of fixed bins can be improved by employing an adaptive scene subdivision to enhance the load balancing.

Furthermore, we accelerate the procedure AreOverlapping (see Algorithm 1 Line 6) by assigning an axis-aligned bounding box to each projected scene triangle. By quantizing the box coordinates we compress the full description into 32 bits. A fast rejection of non-overlapping triangles can be executed with just a few compare operations.

Since each thread handles the intersection of an overlapping triangle with the edge that is assigned to its warp, we obtain either two or zero intersections per thread. The final storage address in global memory is the sum of two offsets. A global offset which is acquired on a per-warp basis via global memory atomics and a per-thread offset which is obtained by computing an intra-warp prefix sum of the threads' intersection counts. Sorting the intersections along each edge is achieved by a key-value sort which arranges all intersections according to edge index and position along their respective edge. The edge index (a 32 bit integer) and the position along the edge (a 32 bit float) of each intersection is combined into a 64 bit radix sortable key. We use the radix sort method of the *thrust* library (Hoberock and Bell 2010) in our implementation.

Algorithm 4 Chunked parallel scan.

Input: A list I of arithmetic values.
Output: DoSomething(S, n) receives the n-th chunk of the scanned values S of I.

```
1:  procedure ChunkedScan(I)
2:      CS ← 32                                   ▷ chunk size (here warp size)
3:      c ← 0                                     ▷ carry (in shared memory)
4:      for n ← 1 to ⌈|I|/CS⌉ do
5:          l ← min(CS, |I| − (n − 1)CS)          ▷ last index
6:          for k ← 1 to l do in parallel
7:              V(k) ← I((n − 1)CS + k)
8:          end for
9:          S ← ParallelIntraWarpScan(V, l)
10:         for k ← 1 to l do in parallel
11:             S(k) ← S(k) + c
12:         end for
13:         c ← S(e)
14:         DoSomething(S, n)
15:     end for
16: end procedure
```

Algorithm 5 Analytic filter convolution.

Input: L is the set of all visible line segments (with a reference to their respective triangle). L_τ denotes the subset of L which is relevant for tile τ of the output image. F is the supplied convolution filter.

Output: WriteTile writes a tile of the output image.

```
1:  procedure Integration(L)
2:      for all tile τ do in parallel
3:          for all batch b in L_τ do
4:              for all pixel p ∈ τ do in parallel
5:                  for all segment l ∈ b do
6:                      τ(p) ← Integrate(l, p, F)
7:                  end for
8:              end for
9:          end for
10:         WriteTile(τ)
11:     end for
12: end procedure
```

The core parts of both algorithm 2 and 3 are the initialization, execution and evaluation of ParallelInclusiveScan. For a given edge e with n intersections, the number of values which have to be scanned can be up to n. A first approach would be to store these values in fast shared on-chip memory and execute the scan over the whole array. Shared memory is a very limited resource ($\sim$ 48kB per SM) and thus only a limited amount of values can be stored before the number of warps per SM, which can be launched in parallel, is significantly reduced. Storing the intermediate values in the much larger global memory is prohibitively expensive in terms of memory access times. Our solution is to conduct the inclusive scan in parallel for warp-sized chunks and execute the chunks sequentially as shown by Algorithm 4 This allows the full utilization of the GPU's SIMD parallelism with a small shared memory footprint.

The procedure ReportSegment uses the same scan method to obtain the correct offset in order to write the line segments into the output buffer. As before, the line segments are output in a non-deterministic fashion and we again employ a radix sort to assign the segments to their respective edges. This list of line segments constitutes the output of our analytic visibility method and provides the necessary information to employ an analytic anti-aliasing method.

5.2.4 Analytic Integration

We implemented the prefiltered sampling of Chapter 4 to render the final output image using CUDA. It should be noted that the input to this stage is an unordered list of line segments per tile. An explicit reconstruction of the visible regions is not needed, since

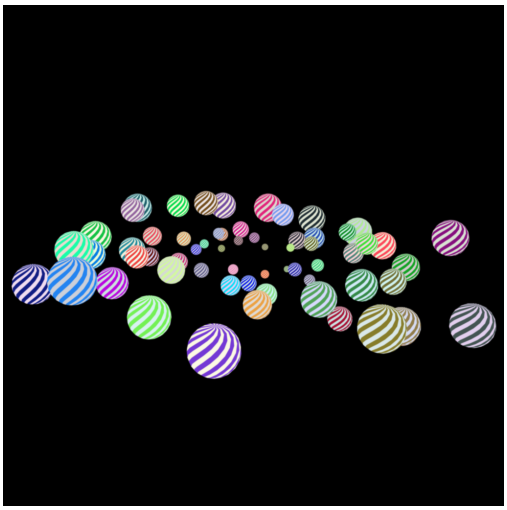
Scene (triangles)	Resolution	Visibility (R/B)			Int (B)	
		128^2	64^2	32^2	8^2	16^2
Bunny (70k)	512^2	✗	155	141	56	236
	1024^2	122	95	96	51	155
	2048^2	83	73	81	71	174
Planets (37k)	512^2	✗	121	111	59	229
	1024^2	94	74	77	31	141
	2048^2	64	56	63	40	100

Table 5.1: Optimal bin size for our implementation. **Vis (R/B)** gives the timings of our analytical visibility method for a bin size **B**, such that **B** $\times$ **Resolution** equals the header value. The timings of the integration stage are given by **Int (B)**, with the header value denoting the bin size. Note that the visibility computation benefits from a certain ration of bin size and resolution, while the integration prefers the smallest bin size.

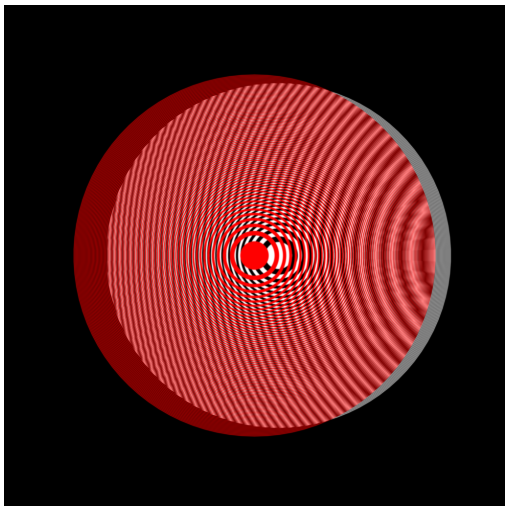
the integration is evaluated over their boundary segments. As shown in algorithm 5, we again employ a two-level parallelization. The output image is subdivided into tiles that are assigned to the warps of the GPU. Each warp alternately executes two distinct stages; in the first stage, each thread loads an input line segment, whereas in the second stage, each thread computes the contribution of all loaded segments to a single pixel of the tile. This reduces the shared memory needs as all input segments reside in the threads' registers, and beginning with the Kepler architecture of NVidida GPUs, register data can be shared by the threads of a warp without shared memory transfers. Once the contributions of all input line segments to the pixels of a tile are computed, the tile is written to the output texture in global memory. Only the assignment of tiles to the warps has to be synchronized, since access to a tile is exclusive for a single warp. As before, we use global memory atomics for this purpose.

5.3 Results

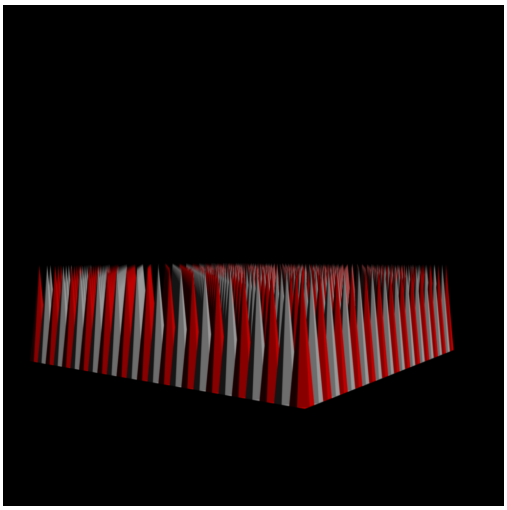
We evaluated the performance of our analytic visibility implementation on a GeForce GTX 680 GPU with 4GB RAM and a Core i7 CPU clocked at 2.67 Ghz with 12GB systen memory. The operating system was Windows 7 with CUDA framework 4.2. Four scenes with different characteristics were used (see Figure 5.5). Zoneplates exhibits very fine scale geometry and serves as a test for the numerical robustness of our method. As can be seen in Figure 5.5b, even geometric intersections of subpixel scale of the two superposed zone plates are correctly resolved. Spikes serves as stress test for a large edge intersection count, due to its high depth complexity. As standard scenes we use a stylized system of Planets and the Stanford Bunny. All scenes were rendered with a Gaussian filter kernel with a radius of 2.3 pixel (see Chapter D).



(a) Planets



(b) ZonePlates



(c) Spikes

(d) Bunny

Figure 5.5: Our test scenes with low (b) & (d) to high (c) depth complexity and low (d) to high (b) & (c) geometrical detail. The images were generated with our method using a Gaussian filter kernel with a radius of 2.3 pixel (see Chapter D) and a resolution of 1024^2. The ZonePlates scene (b) consists of two superposed zone plates while Spikes is a regular grid of square pyramids. All scenes are robustly handled by our method and run at interactive framerates on modern graphics hardware.

5.3.1 Bin Size

As a first step, we investigated the algorithm's behavior with different bin sizes (see Section 5.2.3). The integration stage benefits from localized line segments and executes

Triangle count	Setup	Resolution	Filled Bins	Edge intersections	Hidden-line elimination	Boundary completion	Integration	Overhead	Total
7k	0.04 (3k)	512^2	918 (16k)	6.3 (0.5M)	0.4 (75k)	1.0 (25k)	7.4	6.4	25
		1024^2	3.4k (29k)	9.9 (0.6M)	0.8 (0.1M)	1.6 (33k)	9.1	7.8	34
		2048^2	13k (68k)	21 (1.2M)	1.6 (0.3M)	2.7 (55k)	16	8.8	56
26k	0.16 (13k)	512^2	910 (44k)	31 (1.5M)	1.1 (0.2M)	3.5 (87k)	23	9.4	75
		1024^2	3.4k (63k)	26 (1.9M)	1.6 (0.3M)	4.2 (0.1M)	21	10	71
		2048^2	13k (116k)	37 (2.7M)	2.8 (0.5M)	6.0 (0.1M)	33	13	100
70k	0.42 (30k)	512^2	906 (102k)	116 (3.9M)	2.6 (0.5M)	10 (0.3M)	56	16	212
		1024^2	3.3k (134k)	77 (4.5M)	3.4 (0.7M)	11 (0.3M)	51	18	172
		2048^2	12k (212k)	79 (5.9M)	5.3 (1.0M)	14 (0.4M)	72	22	208

Table 5.2: Rendering statistics from prefiltered rasterization of the Bunny model at different LODs with our method. Timings are given in milliseconds and the size of the output of each stage is given in parentheses. See the text for details.

fastest for the smallest bin size (in our case 8^2 pixel). However, the visibility stage shows the best performance at certain bin sizes relative to the image size, i.e., for a certain ratio between resolution and bin size (see Table 5.1). Smaller bin sizes quadratically increase the number of bins that a given triangle is assigned to. This could cause multiple computation of the same intersection between an edge and a triangle in different bins, thus incurring a performance penalty. For too large bins, the number of triangles per bin increases and causes a quadratic increase in the number of intersection computations. The preferred ratio of the visibility phase for a given resolution is consistent across scenes and can be taken as a performance guideline. Due to the significant increase in computation time of the whole pipeline for bin sizes greater than 8^2, caused by the integration stage, we use bin size 8^2 for the following time measurements.

5.3.2 Timings

Table 5.2 gives a detailed overview of the statistics and timings of our method when rendering different LODs of the Bunny test scene. The first and fourth column show the triangle count of the model and the number of bins that had at least one triangle assigned to. All other columns give the execution time of the respective kernels in milliseconds. The values in brackets is the number of output elements of the column's computation. They are, from left to right, frontfacing triangles, bin-triangle-assignments, edge intersections, visible edge segments, and the line segments to complete the boundaries of the visible regions. The overhead column gives the total runtime of all intermediate radix sorts, scans, and initializations. Column **Total** gives the total time of all GPU operations. The rendering of all LoDs at the given resolutions runs at interactive framerates and it can be seen that most of the runtime of the visibility stage is spent computing the edge intersections.

ZonePlates and Spikes illustrate the robustness of our geometric computations, and in table 5.3 we give an overview of their render timings. As expected, we see that the runtime of the visibility stage depends mainly on the number of generated edge intersections, while the integration procedure depends on the size of its line segment input.

5.3.3 Comparison with Supersampling

An informal comparison with traditional sampling-based rasterization is given in Figure 5.6 using multipass hardware rasterization with DirectX. Although a substantial amount of samples is needed for scenes with high anti-aliasing requirements, traditional sampling still has a runtime advantage over our method.

5.4 Discussion

We have presented an analytic visibility method to perform exact hidden-surface removal on massively parallel hardware architectures. We showed that with an adequate geometric

Scene	Size	Intersection	Segments	Visibility	Integration
Planets	37k	3.7M	576k	94	31
Spikes	5.0k	25M	973k	448	43
ZPlate	14k	4.7M	1.6M	165	104
Bunny	70k	4.5M	941k	122	51

Table 5.3: Rasterization statistics of our test scenes (see Figure 5.5) with our method using a bin size of 8^2 and a resolution of 1024^2. **Size** gives the number of scene triangles, **intersections** the number of edge intersections, and **segments** the number of visible line segments. The timings of the complete **visibility** stage and the analytic **integration** are given in milliseconds. See the text for details.

computation scheme and adaptation to the two-level parallelism of SIMD architectures, it is possible to robustly perform prefiltered anti-aliased rendering of 3D scenes at interactive frame rates on GPUs. A possible future extension of our pipeline can be the use of dynamic load balancing with adaptive scene subdivisions in contrast to our static tiling. Recent developments of GPU architectures (e.g., *Dynamic Parallelism* of NVidia) support such approaches.

While sampling-based rasterization proves hard to beat in terms of speed, we see our work as a first step to bring back analytic methods to rendering. The applications of our method are plenty and largely unexplored. Just employing the hidden-line-elimination stage allows analytic line rendering. The output of our visibility stage gives a full description of the scene visibility from a viewpoint and can be used to generate an analytic visibility map, analytic shadow maps or direct rendering to vector graphics. Furthermore, the method does not depend on the final image resolution and will have advantages for large-scale images. In the integration phase, the polynomial filter function can be altered on the fly, which allows the use of different filters in the same output image. This can be applicable for motion blur or depth-of-field effects. A change in the visibility algorithm could allow analytic depth peeling and support analytic anti-aliased transparency effects.

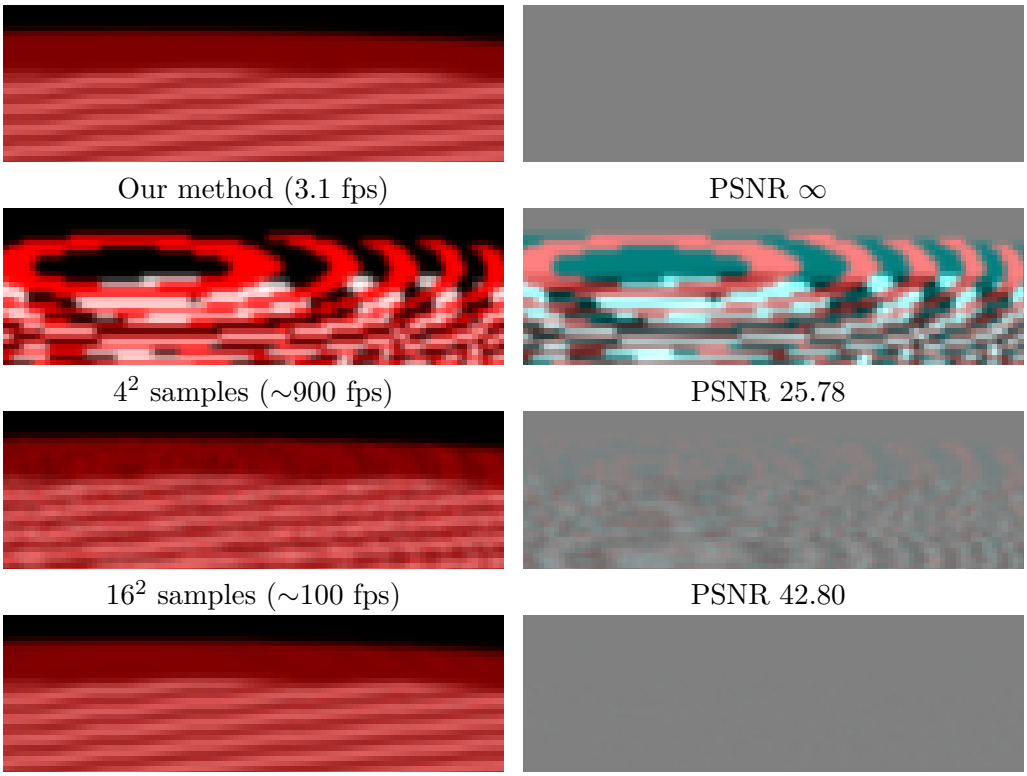


Figure 5.6: Comparison of our method with massive supersampling at a resolution of 1024^2. The left column shows a detailed view of a ZonePlates rendering with our method (top) and with a supersampling approach with three different sampling densities. The sample count is per-pixel and the timings are for the full render cycle. The right column gives the corresponding difference images and the peak signal-to-noise ratio (PSNR) when compared with our rendering. The supersampling method uses stratified sampling and sample sharing by collecting the samples over multiple rendering passes. Note that for highly detailed models, 16^2 samples are not sufficient to faithfully approximate the Gaussian filter kernel that is evaluated analytically with our method. A break-even in terms of fps with our method is reached for approximately 100^2 samples and the reference solution with 128^2 samples gives near-identical results.

CHAPTER 6

Non-Sampled Anti-Aliasing

In the previous Chapters 4 and 5, prefiltered sampling was introduced for different scenarios. The two-dimensional rasterization (resp. three-dimensional voxelization) technique of Chapter 4 was augmented with analytic visibility in Chapter 5 to allow fully prefiltered 3D-to-2D rasterization. A major drawback of exact prefiltering is the requirement of closed-form solutions for the convolution integrals of the filtering process. For perspective-correct shading or general non-linear illumination models, an analytic expression of the solution is rarely obtainable. In this chapter, we will demonstrate how sampled shading can be combined with prefiltered visibility to provide the best of both worlds—near-perfect edge anti-aliasing and general shading.

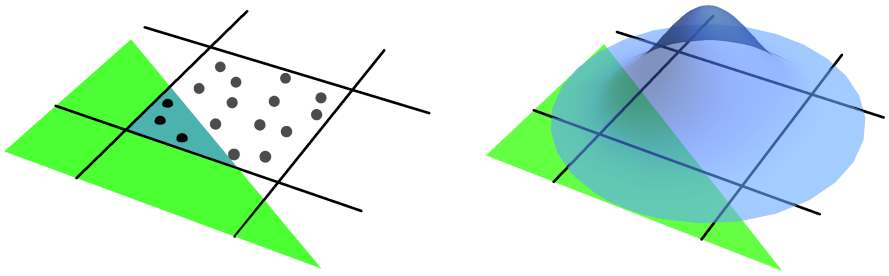


Figure 6.1: Multisampling methods supersample the visibility function of the scene primitives and determine the contribution to the output pixel by the ratio of primitive hits (left). Our optimal edge anti-aliasing method computes this contribution with an exact integral of a filter kernel with the visible area of the primitive (right).

6.1 Motivation

There are two main sources of aliasing artifacts in rasterization: (1) undersampling of the interaction of light with surface materials and (2) undersampling of the visibility of the object geometry. This is a consequence of the fact that the scene data is discretized by the rasterizer before shading or visibility computations are executed. Artifacts caused by (1) are, for example, low-frequency patterns in distant textures or missing highlights. (2) arises under various circumstances, e.g., along edges at silhouettes or creases of objects, very small objects that disappear between the samples, or complicated detail where features are lost (Crow 1977). This is commonly referred to as *edge aliasing* and is an active research area in computer graphics since decades.

To combat these defects, different anti-aliasing methods can be applied separately to both channels. In rasterization, Supersample Anti-Aliasing (SSAA) usually refers to identical supersampling of both, while Multisample Anti-Aliasing (MSAA) supersamples only the visibility. We will follow the latter example and employ high-quality anti-aliasing on the visibility signal and rely on common sampling for the shading channel. Instead of supersampling, we build on the analytic visibility methodology that we developed in Chapter 5 and present with Non-Sampled Anti-Aliasing (NSAA) a technique for prefiltered edge anti-aliasing (see Figure 6.1). This is generally possible, since the visibility signal is a piecewise constant function, where the discontinuities are piecewise linear curves. Thus, a closed form solution of such a signal's convolution with an approximated radial filter always exists. In contrast to shading prefiltering as presented in Chapter 4, we do not pose any restrictions on the shading signal.

6.2 Non-Sampled Anti-Aliasing

Our aim is to produce a perfectly edge anti-aliased raster image by analytically solving the convolution

$$w_P = \int \chi_P(\boldsymbol{x}) W(\boldsymbol{s} - \boldsymbol{x})\, d\boldsymbol{x} \tag{6.1}$$

of the *visible* projected area of a primitive P (given by its characteristic function χ_P) with a prefilter W at a given sample location $\boldsymbol{s}$ (usually a pixel center). The weight w_P is then used to blend the shading samples at s. In multisampling terminology, this equates to taking an infinite amount of visibility samples $\boldsymbol{x}_i$ around $\boldsymbol{s}$, and using their weighted sum $\sum_i W(\boldsymbol{x}_i - \boldsymbol{s})$ for the final blending. Common multisampling techniques restrict their sample positions to a rectangular pixel region, whereas our method applies a radially symmetric prefilter of arbitrary extent, thereby allowing for unbiased sampling of the image signal.

Our proposed analytic pipeline consists of three main stages, depicted in Figure 6.2:

Primitive gathering: first, the relevant primitives that have to be considered for the prefiltering at each sample location are listed. These are all primitives that intersect the support of the filter kernel when placed at the individual pixel centers.

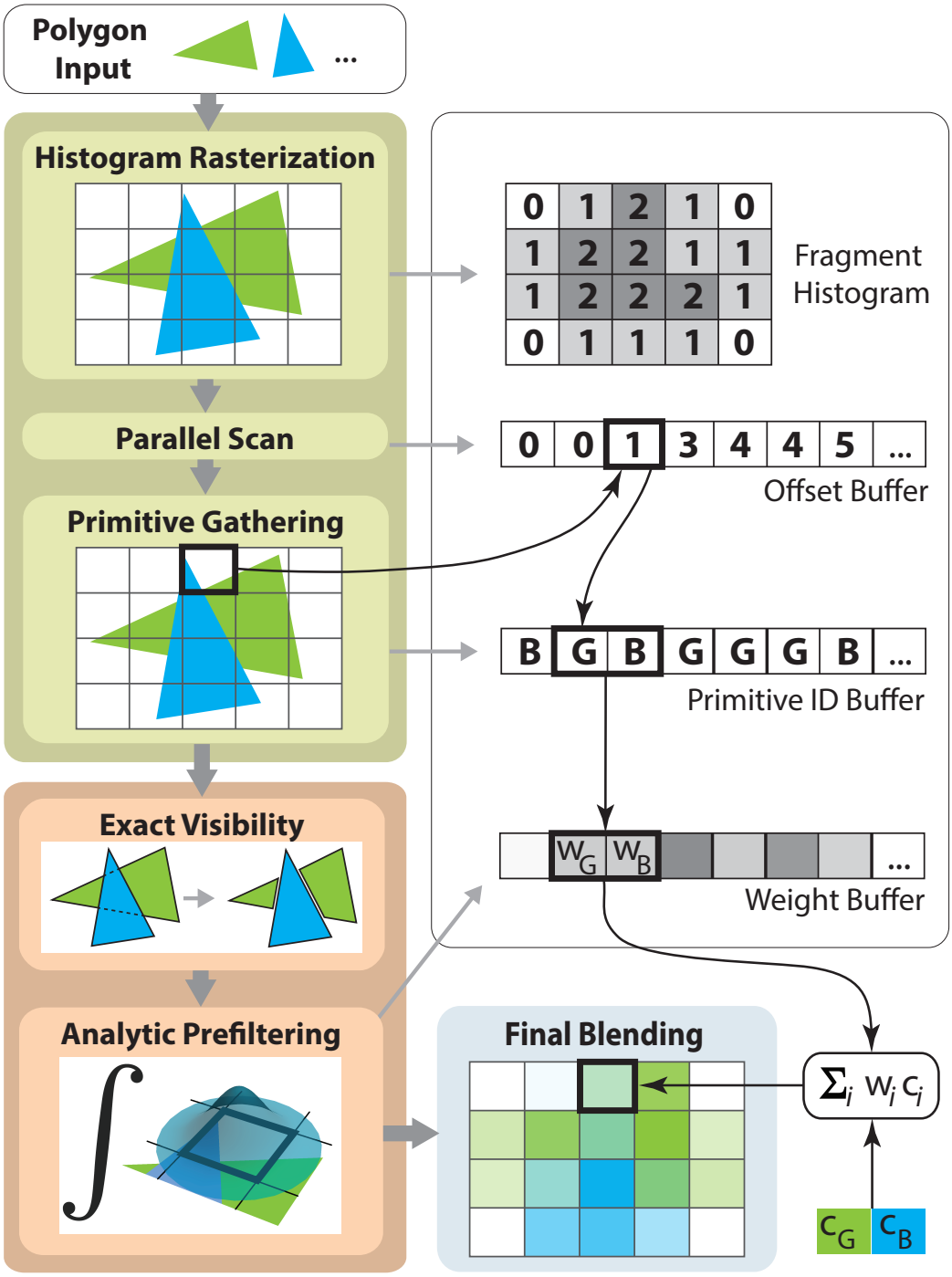


Figure 6.2: Overview of our NSAA pipeline and its three main stages: primitive gathering (green, Section 6.2.1), weight computation (red, Section 6.2.2), and final blending (blue, Section 6.2.3). Intermediate buffers are shown in gray.

Weight computation: given the list of relevant primitives per sample, their contribution to the final sample color is computed by analytically evaluating the convolution given by equation 6.1. These weights are used in the subsequent blending stage.

Blending: the primitives are rasterized and the fragments are blended with weights of the previous stage.

Our framework was developed with current graphics architecture in mind and efficiently utilizes the rasterization pipeline as well as the General-Purpose Computing on Graphics Processing Units (GPGPU) capabilities of modern Graphics Processing Units (GPUs). Conceptually, our framework can accommodate arbitrary 2D primitives and every regular spacing of sample locations. In our implementation, however, we assume triangles as input and pixel centers as our sample locations. The individual stages of our pipeline, as shown in Figure 6.2, are described in detail in the following sections.

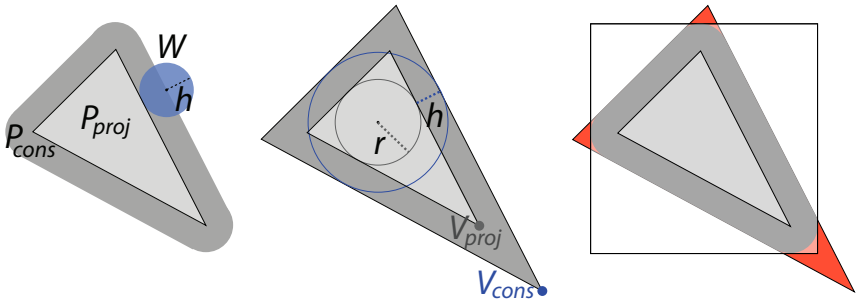
6.2.1 Primitive Gathering

Convolution can be imagined as placing a prefilter at each sample location and evaluating the corresponding integral with each primitive (see Equation 6.1). Mathematically, such a filter function should have infinite support to provide the best possible low pass performance and it would rapidly vanish with increasing distance from its center. As numerical precision is limited and minor changes in a filter are not visually perceivable, it is safe to place a cutoff at a certain distance from the center. This way, the influence region of the filter kernel around each sample location is limited and only close primitives are relevant. This considerably reduces the workload for large scenes or large output images as otherwise the number of possible pairings would grow with the product of primitive and sample count.

Given a set of input primitives, the first task of our algorithm is to list all the primitives that intersect the filter kernel of a given sample. As the number of intersecting primitives per sample can vary strongly depending on the scene, it is beneficial to store the per sample primitive lists in a linear, tightly packed buffer (see Figure 6.2). This *primitive ID buffer* is computed in two steps:

List Offset Computation: First, the required list sizes for each sample are computed. This is easily achieved by an initial scene rasterization pass, that accumulates a histogram of intersecting primitives using additive blending. In a following GPGPU pass, a *parallel scan* (Hoberock and Bell 2010) over the linearized histogram buffer is performed to obtain a *list offset buffer*, which contains the index of each samples' first list element in the primitive ID buffer.

Filling the Primitive ID Lists: In a second scene rasterization pass, the list offsets are used to scatter the primitive ID of each primitive to the corresponding positions in the primitive ID buffer. As the histogram count above, this can be performed conflict-free for concurrently writing threads by employing atomic counters. For



(a) (b) (c)

Figure 6.3: (a) For a given projected primitive P_{proj} we need to rasterize the conservative area P_{cons} in order to address all placements of the prefilter kernel where it intersects P_{proj}. (b) A triangle is conservatively enlarged by scaling it around its incenter according to its incircle radius r and the kernel radius h. (c) During rasterization, all dark red locations outside the minimum conservative box are discarded to reduce the number primitive IDs whose convolution evaluates to zero as they lie outside of P_{cons}. Note that the light red region still reports superfluous IDs but they are negligible when compared to the triangle area.

this task, we use an additional count buffer of the same length as the offset buffer. The buffer index of each primitive ID is then given as the sum of the offset and the count. The resulting primitive ID buffer is then transfered to the next stage for analytic visibility and weight computation.

Note that both rasterization passes are performed without shading or depth buffering. All primitive IDs are collected and visibility is computed at the next stage.

Conservative Rasterization. Due to the finite extent of our radial prefilters, primitives can be relevant for a given sample location without overlapping it, but intersecting only the prefilter support. We therefore compute a *conservative* coverage of all relevant sample locations when rasterizing a primitive during the histogram, primitive gathering, and the final blending stage. Similar to Hasselgren et al. (2005), this is achieved by an image-space thickening of the primitives according to the kernel radius (see Figure 6.3a): in each rasterization pass, we make use of the geometry shader stage to first obtain the projection P_{proj} of an input triangle P, and then compute the required vertex offsets that produce a conservative triangle P_{cons}. Offsetting an image space vertex V_{proj} to its new conservative position V_{cons} basically equates to scaling its corresponding triangle around the center of its incircle by a factor of $\frac{r+h}{r}$, where r is the radius of the incircle, and h denotes the support radius of the filter kernel (see Figure 6.3b). The resulting enlarged primitive P_{cons} is then rasterized, and all its conservative fragments are processed accordingly. To reduce the number of unnecessarily evaluated fragments, which can

become significant for acute triangles, we discard any fragments outside a conservative triangle clipping box in the pixel shader (see Figure 6.3c).

6.2.2 Analytic Weight Computation

In this stage we compute the exact contribution of all primitives to the given sample locations. As shown in Figure 6.2, it consists of two main steps. The analytic visibility step performs hidden-surface elimination and outputs the visible parts of all primitives for the given view direction. This constitutes the exact visibility signal on which we apply prefiltering. Note that in traditional depth-buffering the visibility signal is (super)sampled directly, whereas we rely on a closed-form solution of the filter convolution integral to avoid any undersampling issues. Assuming triangular primitives, the visible regions are polygons and the convolution can be decomposed into a sum of integrals over the linear boundary segments of each polygon. For both the visibility and the prefiltering step we employ previous works and a summary of both is provided in the next paragraphs.

The hidden-surface elimination is performed with the help of the analytic visibility method of Chapter 5. The output of this step is a set of line segments which constitute the boundaries of all visible regions of all scene primitives. Note that this step does not depend on the output of the primitive gathering section of our pipeline and can run concurrently with it or on another device.

In the following prefiltering step, we exploit the geometric simplicity of the polygonal visible regions when evaluating integrals over them. Previous works on analytic filter convolution, such as the method of Manson and Schaefer (2013) and our method of Chapter 4, decompose the convolution over each visible region (see Equation 6.1) into a sum of integrals over its boundary. Furthermore, they support linear shading functions and, thus, include our case of the constant characteristic function χ_P. Each summand of the convolution decomposition can be expressed in closed form and evaluated with the numerical precision of the hardware.

In contrast to these works, we do not employ full analytic shading as advanced shading effects cannot be solved in closed form. We use the two methods of Chapter 4 and 5 just for visibility prefiltering. This can be seen as a substitution for the depth-buffering in the traditional 3D graphics pipeline. Hidden-surface elimination is performed once per frame and yields the aforementioned set of boundary line segments that is used as input to the prefiltering step together with the primitive ID buffer. As each buffer entry id is associated with a sample location $\boldsymbol{s}$, we store the result of the filter convolution $\int \chi_{id}(\boldsymbol{x})W(\boldsymbol{s}-\boldsymbol{x})\,d\boldsymbol{x}$ at the corresponding entry of the weight buffer. This can be executed in parallel for all entries of the primitive ID buffer and allows efficient utilization of massively parallel hardware architectures. The filled weight buffer constitute the output of this step.

6.2.3 Final Blending

For each sample location s and all its corresponding primitives P_i, the previous stage outputs an analytically computed weight w_i that encodes the primitive's relative con-

tribution to the final color of the sample (see Figure 6.2). We are left to compute a *shading value* c_i for each primitive. In traditional rasterization, shading is performed directly at the sample location (usually the pixel center or the centroid of contributing subsamples). Due to the positive extent of our prefilter, it cannot be guaranteed that the projected primitive overlaps the sample location. In this case we project the sample location onto the primitive, i.e., we take the point on the primitive with minimal distance to the sample location as our shading location.

By issuing another rasterization pass, we can now compute the shading value c_i at this point, for example via texturing or non-linear illumination and interpolation models. As each sample location gets contributions from various primitives, we employ conventional blend operations to aggregate the final sample color. Using a normalized prefilter kernel ensures that $\sum_i w_i \leq 1$ for all weights w_i associated with a sample location. $1 - \sum_i w_i$ gives the weight of the background and is non-zero if the support of the local prefilter is not completely covered by the projected scene primitives. Thus, the final color of a sample is given by $\sum_i w_i c_i + (1 - \sum_i w_i) c_B$ where c_B denotes the background color. Note that no depth buffering is used in this final rasterization step and that the scene visibility was already resolved by the weight computation stage which assigns zero weight to occluded primitives.

In this work we use a single shading value per primitive and sampling location. Supersampling of the shading channel is a promising future extension of our framework for which the optimal sample positioning poses the biggest challenge. Current multisampling approaches place the shading sample inside the projected area of the primitive whereas the correct solution would require a placement inside the *visible* projected area. Otherwise it is possible to obtain shading values from hidden parts of the scene. For general triangular scenes, the visible regions of a single input primitive can already consist of several non-convex polygons and an efficient sampling of these regions would deliver the optimal anti-aliasing quality for rasterization. This would require fundamental changes to our pipeline, however, since the shading sample locations do not coincide any longer with the rasterization grid of common graphics hardware.

6.3 Results and Applications

We developed a prototype implementation of our method using the graphics Application Programming Interface (API) DirectX 11 and the GPGPU framework Compute Unified Device Architecture (CUDA). The rasterization steps in both the *primitive gathering* and *final blending* stage of our pipeline use traditional rasterization with deactivated depth buffering. The parallel scan and the *weight computation* stage rely on the flexibility and parallel performance of general-purpose graphics hardware. In the following sections we provide an evaluation of the anti-aliasing performance of our implementation.

(a) MSAA 1 (b) MSAA 4 (c) CSAA 32Q (d) SMAA (e) NSAA (our method)

Figure 6.4: Comparison of common anti-aliasing techniques with our method NSAA using a Gaussian prefilter. The second row depicts a cutout of the top row zoneplate test pattern that was rasterized using the stated method. Note that the performance of anti-aliasing methods is tied to the associated sampling characteristics. If the sampling is changed after applying anti-aliasing, artifacts in the form of excessive blurring or aliasing are introduced. We refer the reader to the electronic version of this paper to inspect the top row pattern close to their native resolution of 1024^2. The cutouts are added as a convenience for readers of the printed version.

Figure 6.5: Comparison of common anti-aliasing techniques with our method NSAA using a Gaussian prefilter. The second row depicts a cutout of the top row log grid test pattern that was rasterized using the stated method. See Figure 6.4 for a further explanation of the two rows.

Figure 6.6: Comparison of different prefilters. Since our method provides an exact evaluation of the filter convolution with the visibility signal, our results can be used to directly compare prefilters on general scenes. A comparison with CSAA 32Q is given to show that aliasing artifacts of common multisampling techniques would dominate the effects of different filter functions. We show the results for three radially symmetric filters each with a radius of two pixel: a constant area filter, a cone shaped tent filter and a Gaussian. See Figure 6.4 for a further explanation of the two rows.

6.3.1 Evaluation

We use well established test patterns to asses the quality of our method. The first pattern is a zoneplate, which features concentric rings with decreasing width to introduce increasing frequencies at the outer rings. The result of our method using a Gaussian filter with a radius of two pixels is illustrated in Figure 6.4 top right. The color pattern is applied with texture mapping. As one can easily see, the result is free of undersampling artifacts due to the exact prefiltering even at the demanding outer rim. Furthermore, the use of radial filters avoids the introduction of anisotropy artifacts along the image diagonal.

As a second test case we employ a binary log grid to investigate the anti-aliasing quality of straight lines. Using the same prefilter as above, we notice no perceivable artifacts in the result shown in Figure 6.5 top right.

A simple test scene in Figure 6.7 shows the correct occlusion handling and a non-linear shading model that has no closed-form solution. By restricting our analytic prefiltering to the visibility signal, sample-based shading models such as texture mapping are supported by our framework.

6.3.2 Ground-Truth Generation

One possible application of our method is the objective comparison of approximative anti-aliasing methods. In the first and second row of Figures 6.4 and 6.5 we compare our output with widely used multi-sampling methods. MSAA1 to MSAA4 are sampling pattern that are specified by both the Open Graphics Library (OpenGL) and DirectX standards, while CSAA 32Q is the highest quality version of a NVidia specific variant. We also compare with Enhanced Subpixel Morphological Anti-Aliasing (SMAA), the post-processing method of Jimenez et al. (2012), which operates on the output image of a rendering system. As can be seen, all methods show various kinds of undersampling artifacts in the outer regions of the zoneplate or along the straight lines. As our method provides an analytically obtained reference, it can be considered as ground truth and any divergence from it can be quantified with a suitable error metric.

6.3.3 Performance

In this section we provide an overview of the performance characteristics of our method. For the presented test patterns in Figures 6.4-6.6, which consist of up to 20k triangles, we obtain near-interactive frame rates on a GeForce 680 with 4GB device memory. As expected, the weight computation stage is the bottleneck of our pipeline. More than 90% of the computation time is spent on the exact visibility and the analytic prefiltering, whereas less than 5% of the time is consumed by the rasterization-based sections of the pipeline. Due to the necessity of storing IDs and weights for all primitive fragments, the memory requirement are in the 100MB range for the aforementioned test scenes. A future extension of our framework that we are planning to develop is a tile-based rendering

approach. For complex scenes, both the increased amount of geometry and textures would limit the available memory for our method. Using an adaptive screen-space tiling, depending on the available memory, would expand the capabilities of our framework to rasterize nearly arbitrary scenes. Even when processing only parts of a scene, we expect an adequate saturation of the processing elements of the graphics hardware.

6.4 Discussion

We presented an analytic prefiltering framework to perform exact edge anti-aliasing with polynomial filter functions. An implementation on graphics hardware was described and its evaluation given. Furthermore, we showed the capability of our system to serve as a ground truth for the evaluation of sample-based anti-aliasing methods.

The main limitations of our work stem from the employed analytic visibility method, which does not support intersecting primitives and requires consistently orientated input. This is also the most promising avenue for future work, as the analytic visibility method could be generalized to accommodate not only intersecting triangles but also transparency effects. The framework can also be extended to allow for separable filters in order to give a complete framework for the evaluation of prefilter performances.

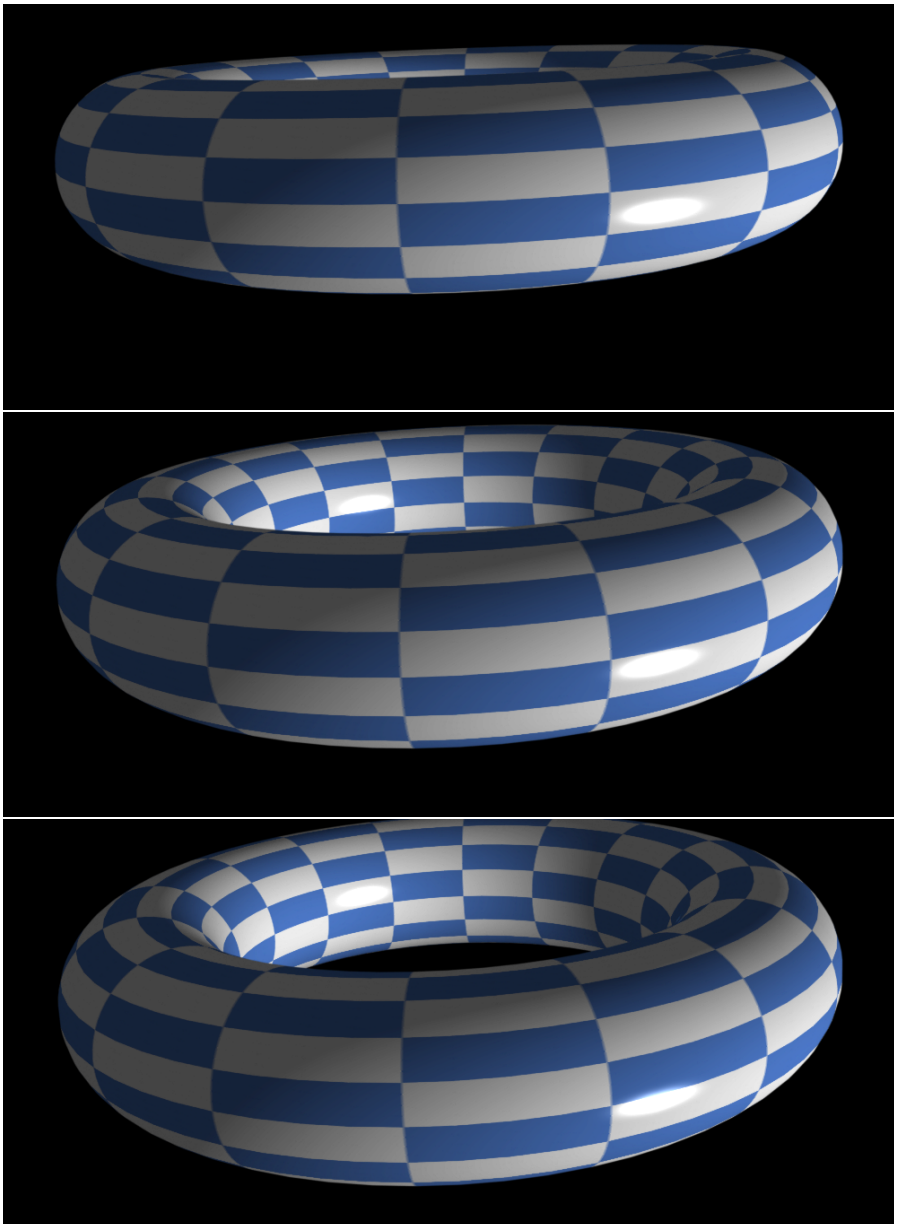


Figure 6.7: A sequence of torus models rasterized with our technique using a Gaussian prefilter of a radius of two pixels. Note that we correctly handle self occlusions of the torus geometry without using any depth buffering methodology. The silhouette exhibits high-quality anti-aliasing and we support general shading models such as texture mapping and Phong shading.

CHAPTER 7

Conclusion

In this chapter, the thesis is concluded and after discussing the main contributions, possible avenues of future research are outlined.

7.1 Summary

This thesis presented different approaches to the effective sampling and filtering of visibility and shading modalities. The foundations of these tasks—commonly referred to as anti-aliasing—were given in Chapter 2 from a signal-processing point of view. There, we also showed that such tasks are integral components of each image-generation process. The main filtering approaches—supersampling and prefiltering—were explained, and their requirements as well as their quality/speed trade-offs discussed. Furthermore, this chapter provided the context of this thesis by surveying related work in this field.

In Chapter 3, we presented a sampling-based visibility-resolution method for view-aligned surfaces. By employing a customized depth function, similar to Generalized Voronoi Diagrams (GVDs), we automatically created cut-aways to reveal relevant patches of otherwise occluded surfaces. We utilized this approach to support blood-vessel inspection tasks, where the surfaces are generated by extrusions away from the centerlines of the vessels. To guarantee well-behaved surfaces away from the centerlines, we presented a Level of Detail (LOD)-based approach that gradually smooths the surface with increasing distance from the centerline. This enables the efficient inspection of surrounding tissues. In this approach, we chose sampling-based filtering methodologies due to the application requirements. Interactivity is vital for medical inspection tasks, while at the same time, our silhouette enhancements do not require any visibility anti-aliasing. This allowed us to omit filtering methodologies and we resolved visibility in a computationally efficient manner by direct ray casting. Furthermore, medical imaging data has limited sharpness and simple supersampling-based shading filtering is sufficient. Remaining aliasing artifacts are then combated with image filtering as a post-process. The interactive performance

and the general usefulness of our approach was documented with measurements and an evaluation by domain experts.

In the following chapters, we investigated anti-aliasing methodologies that provide maximal quality. By performing convolution filtering in the *continuous* domain—also referred to as prefiltering—we effectively removed aliasing artifacts and provided exact solutions only limited by the numerical precision of the underlying hardware.

A framework to perform prefiltering of linear functions on polytopes in n dimensions was presented in Chapter 4. By closely approximating radial filter functions, we computed closed-form solutions of the convolution integrals that arise during filtering. This was achieved by integration-domain composition, which decomposes the otherwise complex integrations into a sum of integrals on well-behaved domains. We provided the exact mathematical expression in the case of two and three dimensions and presented two-dimensional rasterization and three-dimensional voxelization as applications. While computationally more expensive than sampling-based approaches, we provide ground-truth solutions of shading anti-aliasing.

In Chapter 5, we extended the previous prefiltering method with exact hidden-surface elimination. For this, all occluded parts of a three-dimensional scene consisting of triangular meshes were removed. By determining the line segments that represent the boundaries of visible regions of each triangle, we generated a vector representation of the scene visibility. These segments served as input to the aforementioned exact shading anti-aliasing method. Together, these methods perform full visibility and shading prefiltering and generate ground-truth data for 3D-to-2D rasterization. The core part of this method is a sequence of three highly parallel algorithms that robustly solve the geometrical computations.

A common limitation to prefiltering is the requirement for closed-form solutions of the filter-convolution integrals. For general shading functions, such solutions cannot be achieved, and as a remedy we provided in Chapter 6 a method to combine visibility prefiltering with sampled shading. This approach provides exact edge anti-aliasing while still enabling the use of non-linear interpolation or complex illumination models. This was achieved by replacing the depth-buffer visibility resolution stage of a common graphics pipeline with our aforementioned hidden-surface elimination. With this approach, we provided a ground-truth solution to edge anti-aliasing, which can be used as an objective reference for the multitude of anti-aliasing approximations that exist in the literature. Furthermore, we allow the effective comparison of different filter functions.

Like many computational problems in graphics, all our aforementioned methods exhibit a high degree of parallelism, and we make use of this fact by mapping them to massively parallel hardware architectures in the form of Graphics Processing Units (GPUs). While traditional graphics pipelines utilize some form of graphics interface to access the capabilities of the underlying hardware, our approaches do not fit this model. Consequently, we utilized General-Purpose Computing on Graphics Processing Units (GPGPU) methodologies, which allows computation of general problems on graphics hardware. An

introduction into this hardware design and the necessary considerations to unlock its full potential were given in Chapter 2, as well as an overview of related works. In Chapter 5, we provided an in-depth analysis of the parallelization potentials and the mapping to graphics hardware.

7.2 Discussion

Since we treated the anti-aliasing problem from different aspects in this thesis, we feel confident in giving a general comparison of sample-based and prefiltering methodologies. In applied settings, we do not believe that supersampling and its variants will be replaced any time soon. The considerably higher computational complexity of prefiltering only pays off in demanding scenes with a significant amount of high-frequency content. Furthermore, as prefiltering requires the existence of closed-form solutions of the convolution integrals, polynomial integrands are almost always necessary. While we employed global polynomial approximations of the filter kernels, Manson and Schaefer (2013) decomposed the kernels into bicubic patches. Additionally, their work utilizes polynomial curves as shape boundaries, which is mainly relevant for two-dimensional rasterization. Together with our works, this already covers most of the integrands that have closed-form solutions and in this sense, we consider the problem of prefiltered anti-aliasing largely solved. We see its most promising area of application in adaptive anti-aliasing, where only the most demanding parts of a scene are anti-aliased with prefiltering, while the remainder is treated with common sampling-based approaches.

7.3 Future Work

There are several possible avenues to extend the works presented in this thesis.

While we believe that radially symmetric filters are perceptually superior, our prefiltering methodologies can be extended to separable filters, which generally do not exhibit radial symmetry. This would encompass most filter functions that are used in convolution settings and would allow the ground-truth generation for most anti-aliasing settings.

A further extension in this spirit is the combination of visibility prefiltering with *super-sampled* shading. Such a method would provide the highest possible rasterization quality while still permitting the use of arbitrary interpolation and illumination models.

While prefiltering approaches are inherently computationally more demanding than sample-based variants, it could be beneficial to just employ them on a sparse subset of the scene. The work of Barringer and Akenine-Möller (2013) shows potential in this regard.

For anti-aliasing quality of arbitrarily high quality, our pipeline could also be implemented with arbitrary-precision arithmetics and exact geometric computations. Such an effort has most likely only theoretical relevance.

Our exact hidden-surface-elimination method has several extension possibilities. By performing a more elaborate analysis of the edge intersections among triangles, the depth ordering of several triangle layers can be established analytically, allowing for exact transparency effects. An extension to ground-truth shadow mapping would require the incorporation of the shadow edges into the analysis. The intermediate result can also be reported as vector graphics; however, a reconstruction of the visible regions from their unordered boundary segments has to be performed.

APPENDIX A

Questionnaire of Curved Surface Reformation

In this questionnaire, several aspects of the proposed visualization technique Curved Surface Reformation (CSR) will be investigated and evaluated. The method will be compared to existing and well established visualization approaches such as Curved Planar Reformation (CPR), Multipath Curved Planar Reformation (mpCPR) and Centerline Reformation (CR). Throughout this questionnaire please give a **single** or **no** answer for each question in the ☐ field, i.e., no question allows multiple answers. If you want to correct an answer, please indicate this explicitly and tick your new answer. For several questions it would be highly beneficial for us if you could elaborate on the reasons for your answer; these questions are marked with the following text: *(please justify your decision in the comments)*. By doing so you allow us to benefit from your expert opinion, enhance this evaluation and provide us essential information for future improvements of relevant software tools. In case of any other comments, please feel free to write them into the provided comment field as well. All comments are very appreciated and will help us to strengthen this evaluation.

A.1 General Assessment

In this section, general aspects of the three methods (CPR, CR and CSR) should be anonymously evaluated. Several images are shown on the next page, without knowing which particular technique has been used.

Vessel Lumen — images (a)-(c)

*The **vessel lumen** is defined as the cut through the vessel along its centerline which covers the full width of the vessel. As this is the most important region to assess pathologies, e.g., calcifications, the visualization technique has to depict it as precise as possible.*

1	Which image depicts the vessels best?	a☐	b☐	c☐
2	Which image shows the flow channels qualitatively better?	a☐	b☐	c☐
			none☐	

Surrounding Parts — images (a)-(c)

*The **surrounding parts** of a vessel are the tissues and organs around the vessel lumen. They show additional information that might be useful for spotting pathologies not necessarily related to the vessels themselves. Some CPRs show artifacts in these regions. Open questions such as if the surrounding parts are desired or if they should smoothly extend to the borders of the image are investigated here.*

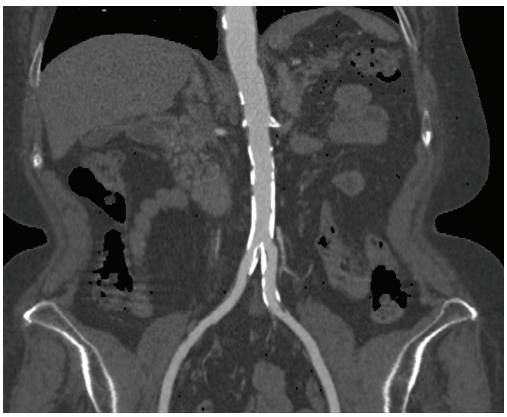
3	Which image shows the organs around the vessels with the lowest number of artifacts?	a☐	b☐	c☐
4	Is the visualization of the organs around the vessels desired? *(please justify your decision in the comments)*	YES☐		NO☐
5	Is the visualization of the organs around the vessels helpful? *(please justify your decision in the comments)*	YES☐		NO☐

Visibility — images (d)-(f)

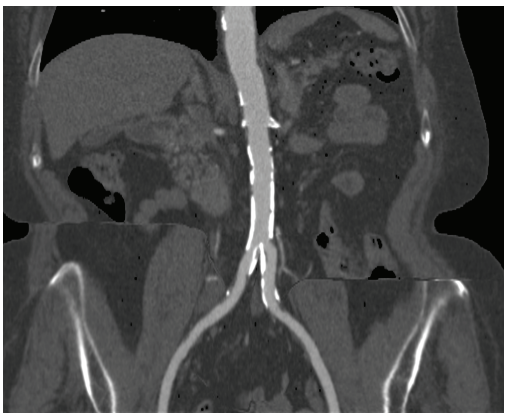
*Finally, the correct **visibility** of the cuts through multiple vessels within one image will be assessed on the example of cervical vessels. Visibility refers to the depiction of correct obstructions of multiple vessels, i.e., which vessel is in front of another one.*

6	Which image has the best visibility of all vessel lumen?	d☐	e☐	f☐
7	Which image preserves the relative location of the vessels to each other best?	d☐	e☐	f☐
8	Is the three dimensional visibility of several vessels in one image desired? *(please justify your decision in the comments)*	YES☐		NO☐
9	Which vessel is closer to the viewer in image (d)?	1☐		2☐
10	Which vessel is closer to the viewer in image (e)?	1☐		2☐
11	Which vessel is closer to the viewer in image (f)?	1☐		2☐

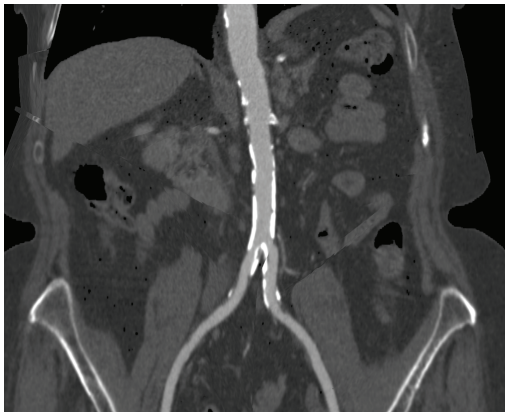
Comments:



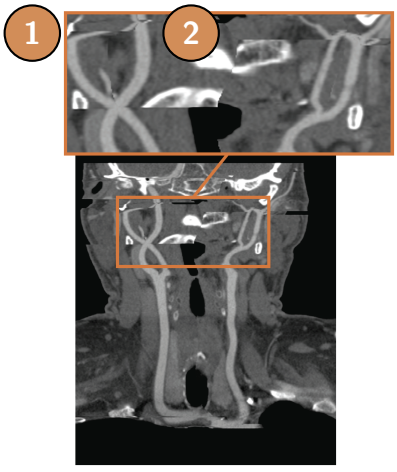
(a)



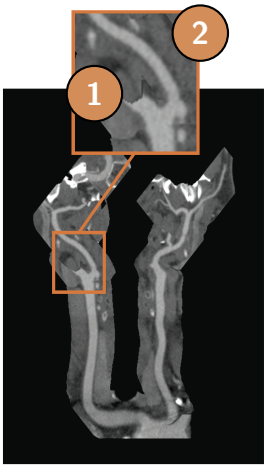
(b)



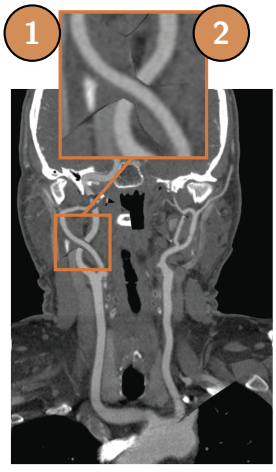
(c)



(d)



(e)



(f)

A.2 Perception

In this section we investigate perceptual issues of various vessel visualization methods. Several images on the next page show the abdominal aorta with its first branches. Your task is to decide, which vessel branch is closer to the viewer (i.e., in front of the other vessel branch).

Questions — images (a)-(f)

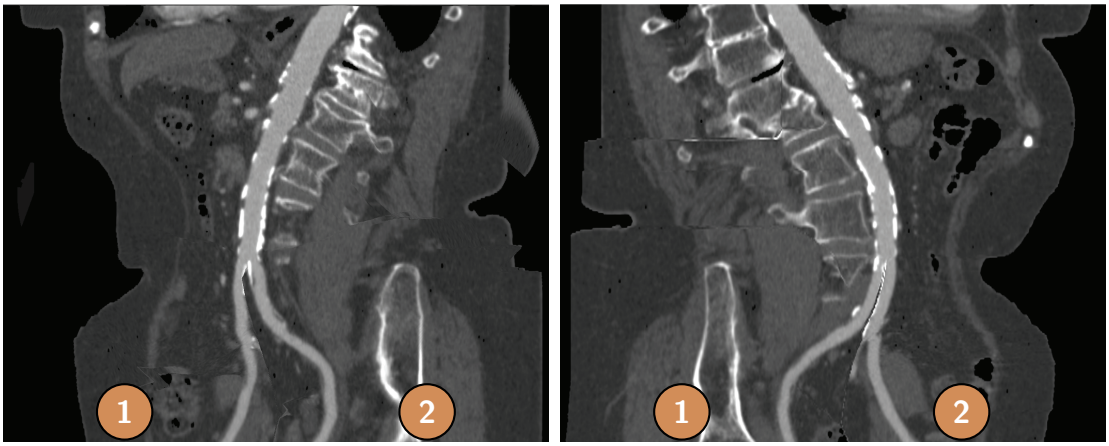
12	Which vessel is closer to the viewer in image (a)?	1☐	2☐
13	Which vessel is closer to the viewer in image (b)?	1☐	2☐
14	Which vessel is closer to the viewer in image (c)?	1☐	2☐
15	Which vessel is closer to the viewer in image (d)?	1☐	2☐
16	Which vessel is closer to the viewer in image (e)?	1☐	2☐
17	Which vessel is closer to the viewer in image (f)?	1☐	2☐

Silhouettes — images (g) and (h)

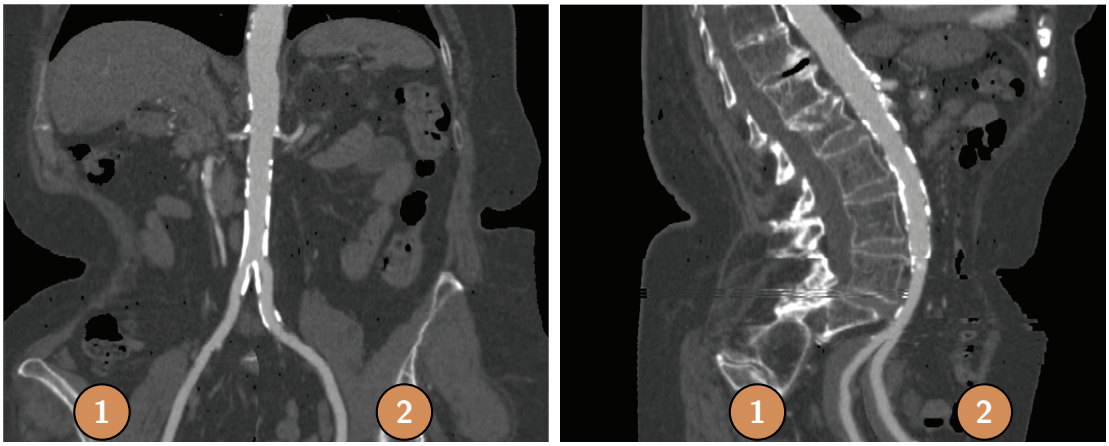
*The **yellow silhouettes** in the images (g) and (h) mark the boundaries between regions with different distances from the viewer.*

18	Which vessel is closer to the viewer in image (g)?	1☐	2☐
19	Which vessel is closer to the viewer in image (h)?	1☐	2☐
20	Are silhouettes helpful for perceiving the vessel orientation in 3D space? *(please justify your decision in the comments)*	YES☐	NO☐
21	Are silhouettes helpful for perceiving the boundaries of regions with different distances from the viewer? *(please justify your decision in the comments)*	YES☐	NO☐

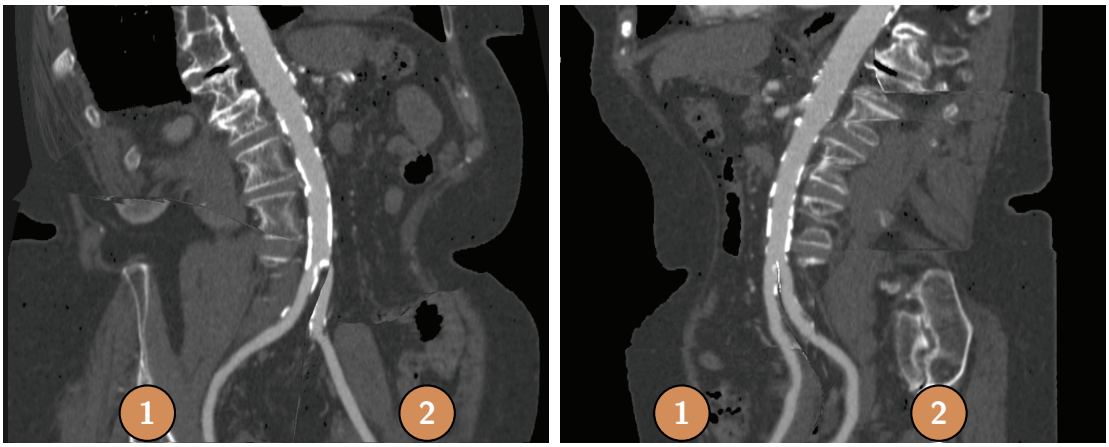
Comments:



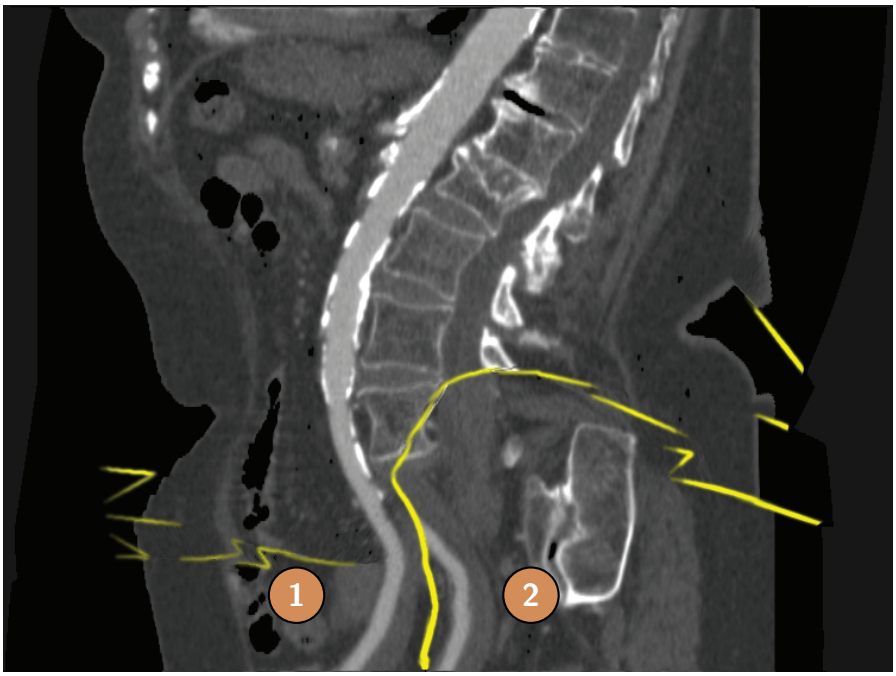
(a) (b)



(c) (d)



(e) (f)



(g)

(h)

APPENDIX B

Ideal Radial Filter Derivation

In this chapter, we derive the representation of an ideal radial filter in the spatial domain. While this derivation is found in various textbooks, the concrete form of Equation 2.7 is not easy to obtain from the literature, since different definitions of the Bessel function and different parametrization of the Fourier transform are used. In the following, we show the relation

$$\widetilde{\Pi_{\odot,\frac{1}{T}}}(\boldsymbol{t}) = J_{\frac{n}{2}}\left(\frac{\pi\|\boldsymbol{t}\|}{T}\right)(2T\|\boldsymbol{t}\|)^{-\frac{n}{2}}. \qquad \text{(2.7 revisited)}$$

In frequency space, the ideal radial filter is given as

$$\Pi_{\odot,\frac{1}{T}}(\boldsymbol{\xi}) = \Pi_{\frac{1}{T}}(\|\boldsymbol{\xi}\|)$$

and we have to compute its n-dimensional inverse Fourier transform, i.e.,

$$\int \Pi_{\frac{1}{T}}(\|\boldsymbol{\xi}\|)\, e^{2\pi i\boldsymbol{\xi}\cdot\boldsymbol{t}}\, d\boldsymbol{\xi} = \int \Pi_{\frac{1}{T}}(\|\boldsymbol{\xi}\|)\, e^{2\pi i\|\boldsymbol{\xi}\|\|\boldsymbol{t}\|\cos\theta}\, d\boldsymbol{\xi}$$

where θ denotes the angle between $\boldsymbol{\xi}$ and $\boldsymbol{t}$. Due to the radial symmetry of the filter function—which is preserved by the Fourier transform—, we can choose $\boldsymbol{t}$ to coincide with the first coordinate axis ϕ_1 and compute the integral in hyperspherical coordinates with $\boldsymbol{\xi} = (r, \phi_1, \ldots, \phi_{n-2})$ and $\boldsymbol{t} = (t, 0, \ldots, 0)$. Recall that the $(n-1)$-dimensional surface area of an n-dimensional hypersphere is given by $\omega_{n-1} = \frac{2\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}\right)}$. We have that

$$\begin{aligned}
&\int \Pi_{\frac{1}{T}}(\|\boldsymbol{\xi}\|)\, e^{2\pi i\|\boldsymbol{\xi}\|\|\boldsymbol{t}\|\cos\theta}\, d\boldsymbol{\xi} = \\
&\quad = \int_0^{\frac{1}{2T}} \int_0^{\pi} \cdots \int_0^{\pi} \int_0^{2\pi} e^{2\pi i r t\cos\phi_1}\; r^{n-1}\sin^{n-2}\phi_1 \sin^{n-3}\phi_2 \cdots \sin\phi_{n-2} \\
&\qquad\qquad d\phi_{n-2}\cdots d\phi_1\, dr
\end{aligned}$$

$$
\begin{aligned}
&= \int_0^{\frac{1}{2T}} \int_0^{\pi} e^{2\pi i r t \cos\phi_1}\, r^{n-1} \sin^{n-2}\phi_1 \int_0^{\pi} \cdots \int_0^{\pi} \int_0^{2\pi} \sin^{n-3}\phi_2 \cdots \sin\phi_{n-2} \\
&\qquad\qquad d\phi_{n-2} \cdots d\phi_1\, dr \\
&= \int_0^{\frac{1}{2T}} \omega_{n-2} \int_0^{\pi} e^{2\pi i r t \cos\phi_1}\, \sin^{n-2}\phi_1\, d\phi_1\, r^{n-1}\, dr = \\
&= \int_0^{\frac{1}{2T}} \frac{(2\pi)^{\frac{n}{2}}}{(2\pi r t)^{\frac{n-2}{2}}} J_{\frac{n-2}{2}}(2\pi r t)\, r^{n-1}\, dr = \int_0^{\frac{1}{2T}} \frac{2\pi r^n t}{(rt)^{\frac{n}{2}}} J_{\frac{n-2}{2}}(2\pi r t)\, dr = \\
&= \frac{J_{\frac{n}{2}}\left(\frac{\pi t}{T}\right)}{(2tT)^{\frac{n}{2}}}.
\end{aligned}
$$

The result is given in terms of the Bessel function of the first kind $J_\nu(x)$ defined as

$$J_\nu(x) = \frac{x^\nu}{(2\pi)^{\nu+1}}\, \omega_{2\nu} \int_0^{\pi} e^{-ix\cos\phi} \sin^{2\nu}\phi\, d\phi,$$

which can be represented by elementary functions for odd multiples of $\frac{1}{2}$, i.e.,

$$J_{\frac{1}{2}}(x) = \sqrt{\frac{2}{\pi x}} \sin x,\ J_{\frac{3}{2}}(x) = \sqrt{\frac{2}{\pi x^3}} \left(\sin x - x\cos x\right), \ldots .$$

APPENDIX C

Filter Convolution in $\mathbb{R}^n$

In this chapter, a general strategy to compute the closed-form solutions of n-dimensional convolution integrals is given. As filters, radially symmetric monomials of the form r^m are assumed, which enable the approximation of arbitrary radial filter kernels with any polynomial scheme. The input signal is given by a linear function $\mathcal{I}$ that is defined inside an n-dimensional polytope $\mathcal{P}$. As stated in Equation 4.3, the integral has to evaluated on the intersection of an n-polytope and an n-hypersphere. Since the intersection is, generally, a complex shape, we split it into a potentially large collection of well-behaved subdomains, for which we can evaluate the integral analytically.

The general procedure consists of a recursive subdivision of the polytope along its hyperplanar facets and subsequent rotations to achieve a simple representation of the associated vertices. Starting with $\mathcal{P}$, we consider each of its $(n-1)$-dimensional facets separately. For each facet, an n-pyramid is constructed with the facet as base and the origin as apex. These pyramids are then rotated such that the normal vectors of their bases coincide with the n-th coordinate axis. Thus, all base vertices of *each* pyramid carry the same value c_n as n-th component, i.e., their form is $(\cdot, \ldots, \cdot, c_n)$. For each pyramid we store the rotation R_n and the base coordinate c_n for later use.

In the next step, we consider the bases of each of the aforementioned pyramids separately. The $(n-1)$-dimensional base of each pyramid—which is an $(n-1)$-dimensional polytope—is again split into its $(n-2)$-dimensional facets. In contrast to before, we now place the apex of the new pyramids at $(0, \ldots, 0, c_n)$, where c_n is the n-th component of the vertices of the current pyramid base. We again use the facets as $(n-2)$-dimensional bases to build $(n-1)$-pyramids together with the apex. Each of these new pyramids is now rotated to align the normal of their bases with the second-to-last coordinate axis, i.e., that the form of the base vertices is $(\cdot, \ldots, \cdot, c_{n-1}, c_n)$. Again, the rotation R_{n-1} and base coordinate c_{n-1} for each pyramid is stored.

Since each recursion step decreases the dimensionality of the associated polytopes by one, we arrive after $(n-1)$ applications of our split and rotation strategy at a collection of triangles, i.e., 2-pyramids. Omitting degenerate input geometries, each of these triangles has a unique sequence of $(n-1)$ rotations R_i and $(n-1)$ base coordinates c_i associated with it. In other words, the vertices of each triangle are given as $(0, 0, c_3, \ldots, c_n)$, $(x_0, c_2, c_3, \ldots, c_n)$, and $(x_1, c_2, c_3, \ldots, c_n)$, where the sequence $(c_2, \ldots, c_n)$ is unique for each triangle.

The convolution integral on the intersection of the original polytope $\mathcal{P}$ and the filter support can be computed as a sum of convolutions on the n-pyramids that are spanned by the triangles and the origin. This follows from the fact that this set of pyramids constitutes a partitioning of $\mathcal{P}$. The rotation sequence of each triangle is used to rotate the linear input function $\mathcal{I}$ to conform the current orientation of the triangle (see Equation 4.4). The sequence of c_is determines the radius of the hypersphere in the hyperplane in which the triangle is situated. Depending on the location of the vertices relative to the hypersphere, n qualitatively different integration domain arise. In hyperspherical coordinates, the associated integrations can be evaluated in closed form for each filter order and we present the case of two and three dimensions below.

C.1 Explicit Solutions in 2D and 3D

The following integration results in two and three dimensions are given for a fixed filter order n. It should be noted that although the results contain non-elementary functions such as the Gauss hypergeometric function ${}_2F_1$ and the Appell hypergeometric function F_1, the terms for a fixed order n can be expressed with elementary functions. We use the abbreviation $f(x)|_{x_0}^{x_1}$ for $f(x_1) - f(x_0)$. The formulas were obtained with the help of a symbolic mathematical software and manually corrected, checked and optimized. The variable names used in the formulas are consistent with Figures 4.2 and 4.3.

C.1.1 2D Integrals

Assuming that $0 < r_0 < R$, $n \in \mathbb{N}$ and interpolation coefficients $\gamma, \lambda, \mu \in \mathbb{R}$ we get with polar coordinates $x = r\cos\varphi, y = r\sin\varphi$

$$\int_0^R r \int_{\varphi_0}^{\varphi_1} (\gamma + \lambda x + \mu y) r^n \, d\varphi \, dr = R^{n+2} \left(\frac{\gamma}{n+2} (\varphi_1 - \varphi_0) + \frac{R}{n+3} (\lambda (\sin\varphi_1 - \sin\varphi_0) - \mu (\cos\varphi_1 - \cos\varphi_0)) \right), \tag{C.1}$$

and with the further assumption that $-\frac{\pi}{2} < \varphi_0, \varphi_1 < \frac{\pi}{2}$ we obtain

$$\int_{\varphi_0}^{\varphi_1} \int_0^{\frac{r_0}{\cos\varphi}} r(\gamma + \lambda x + \mu y) r^n \, dr \, d\varphi = r_0^{n+2} \left(\left(\frac{\gamma}{n+2} + \frac{r_0 \lambda}{n+3} \right) {}_2F_1 \left(\frac{1}{2}, \frac{n+3}{2}; \frac{3}{2}; \sin^2\varphi \right) \sin\varphi + \frac{r_0 \mu}{(n+2)(n+3)} \sec^{n+2}\varphi \right) \Bigg|_{\varphi_0}^{\varphi_1}. \tag{C.2}$$

C.1.2 3D Integrals

Assuming that $0 < d < R$, $n \in \mathbb{N}$, $0 < \theta < \frac{\pi}{2}$ and $\gamma, \lambda, \mu, \tau \in \mathbb{R}$ with spherical coordinates $x = r\cos\varphi\sin\theta, y = r\sin\varphi\sin\theta, z = r\cos\theta$ we get

$$\begin{aligned} \int_{\varphi_0}^{\varphi_1} \int_0^{\arccos\left(\frac{d}{R}\right)} \sin\theta \int_0^{\frac{d}{\cos\theta}} r^2 (\gamma + \lambda x + \mu y + \tau z) r^n \, dr \, d\theta \, d\varphi = \frac{d}{n+4} \Bigg(& \frac{(n+4)\gamma + (n+3)d\tau}{(n+2)(n+3)} \left(R^{n+2} - d^{n+2} \right) (\varphi_1 - \varphi_0) \\ & + 8d^{n+3} \sqrt{1 - \frac{2d}{d+R}} \Bigg(F_1 \left(\frac{1}{2}; n+2, -n; \frac{3}{2}; 1 - \frac{2d}{R+d}, \frac{2d}{R+d} - 1 \right) - 3F_1 \left(\frac{1}{2}; n+3, -n; \frac{3}{2}; 1 - \frac{2d}{R+d}, \frac{2d}{R+d} - 1 \right) \\ & + 2F_1 \left(\frac{1}{2}; n+4, -n; \frac{3}{2}; 1 - \frac{2d}{R+d}, \frac{2d}{R+d} - 1 \right) \Bigg) (\lambda (\sin\varphi_1 - \sin\varphi_0) - \mu (\cos\varphi_1 - \cos\varphi_0)) \Bigg) \end{aligned} \tag{C.3}$$

whereas with the additional conditions $-\frac{\pi}{2} < \varphi_0, \varphi_1 < \frac{\pi}{2}$, $0 < r_0 < R$ and $0 \geq \theta_{\mathrm{C}} = \arccos\left(\frac{d}{R}\right) < \frac{\pi}{2}$ we obtain

$$\begin{aligned}\int_{\varphi_0}^{\varphi_1} \int_{\theta_{\mathrm{C}}}^{\arctan\left(\frac{r_0}{d\cos\varphi}\right)} \sin\theta \int_0^R r^2(\gamma + \lambda x + \mu y + \tau z) r^n \, dr \, d\theta \, d\varphi = \frac{R^{n+3}}{4} \Bigg(-\frac{4\gamma}{n+3} \arctan\left(\frac{\sqrt{2}d\sin\varphi}{\sqrt{2r_0^2 + d^2 + d^2\cos 2\varphi}}\right) \\ + \frac{2R(r_0\tau - d\lambda)}{(n+4)\sqrt{r_0^2+d^2}} \arctan\left(\frac{r_0\tan\varphi}{\sqrt{r_0^2+d^2}}\right) + \frac{2R}{n+4} \arctan\left(\frac{r_0\sec\varphi}{d}\right)(\lambda\sin\varphi - \mu\cos\varphi) + \frac{4\gamma\cos\theta_{\mathrm{C}}}{n+3}\varphi \\ + \frac{R}{n+4}\left(\tau\left(\cos 2\theta_{\mathrm{C}} - 1\right)\varphi + 2\left(\theta_{\mathrm{C}} - \cos\theta_{\mathrm{C}}\sin\theta_{\mathrm{C}}\right)(\mu\cos\varphi - \lambda\sin\varphi)\right)\Bigg)\Bigg|_{\varphi_0}^{\varphi_1}.\end{aligned} \tag{C.4}$$

The evaluation of the integrand on a tetrahedral integration domain proves much harder than the previous cases. This can be attributed to the fact that while the integration domain can be very easily expressed in Cartesian coordinates, the filter monomial r^n is easily described in spherical coordinates. Thus we have not found a result formula that permits the filter order n to be expressed as a variable. Nevertheless it is possible to evaluate the integral for each filter order separately. We show the results for the first few even orders as the odd orders produce lengthy expressions.

$$\begin{aligned}&\int_{\varphi_0}^{\varphi_1} \int_0^{\arctan\left(\frac{r_0}{d\cos\varphi}\right)} \sin\theta \int_0^{\frac{d}{\cos\theta}} r^2(\gamma + \lambda x + \mu y + \tau z) r^n \, dr \, d\theta \, d\varphi = \\ &\begin{cases} \dfrac{dr_0^2}{24}\left((4\gamma + 2r_0\lambda + 3d\tau)\tan\varphi + r_0\sec^2\phi\mu\right)\Big|_{\varphi_0}^{\varphi_1} & n = 0 \\ \dfrac{dr_0^2}{360}\Big(\Big(12\left(3d^2 + r_0^2\right)\gamma + r_0\left(20d^2 + 8r_0^2\right)\lambda + 10d\left(3d^2 + r_0^2\right)\tau\Big)\tan\varphi & \\ \qquad + r_0\sec^2\varphi\left(10d^2\mu + r_0\left(6\gamma + 4r_0\lambda + 5d\tau\right)\tan\varphi\right) + 3r_0^3\sec^4\varphi\mu\Big)\Big|_{\varphi_0}^{\varphi_1} & n = 2 \\ \dfrac{dr_0^2}{5040}\Big(\Big(8\left(45d^4 + 30d^2r_0^2 + 8r_0^4\right)\gamma + 6r_0\left(35d^4 + 28d^2r_0^2 + 8r_0^4\right)\lambda + 7d\left(45d^4 + 30d^2r_0^2 + 8r_0^4\right)\tau\Big)\tan\varphi & \\ \qquad + r_0\sec^2\varphi\Big(105d^4\mu + r_0\Big(8\left(15d^2 + 4r_0^2\right)\gamma + r_0\left(84d^2 + 24r_0^2\right)\lambda + 7d\left(15d^2 + 4r_0^2\right)\tau\Big)\tan\varphi\Big) & n = 4 \\ \qquad + 3r_0^3\sec^4\varphi\left(21d^2\mu + r_0\left(8\gamma 6r_0\lambda + 7d\tau\right)\tan\varphi\right) + 15r_0^5\sec^6\varphi\mu\Big)\Big|_{\varphi_0}^{\varphi_1} & \end{cases}\end{aligned} \tag{C.5}$$

APPENDIX D

Polynomial Filter Approximations

THE coefficients for polynomial fits of various common anti-aliasing filters together with the approximation error are given in Table D.2 and D.3. All filters were normalized to the interval $[0, 1]$. The numbers beside the filter names refer to the original filter width, i.e., the Gaussian ($\sigma = 2^{-1/2}$) is cut off at 2.3 and the Lanczos filter is of radius 2. Note that for 3D filters we use only coefficients for *even* polynomials while the 2D filters can also use odd and thus have a lower degree. The coefficients are computed using a least squares fit with the pre-integrated polynomials shown in table D.1. The pre-integration ensures that the response of our multi-dimensional filters $\mathcal{F}_{3D}$ and $\mathcal{F}_{2D}$ on one-dimensional signals are identical to the response of the original one-dimensional filter $\mathcal{F}$ on the same signal, i.e.,

$$\int\int \mathcal{F}_{3D}(x, y, z)\, dy\, dz = \int \mathcal{F}_{2D}(x, y)\, dy = \mathcal{F}(x).$$

monomial (2D / 3D)	2D pre-integration	3D pre-integration
1	$2\sqrt{1-x^2}$	$\left(1-x^2\right)\pi$
x / x^2	$\sqrt{1-x^2} + x^2 \log\left(\frac{1+\sqrt{1-x^2}}{x}\right)$	$\left(1-x^4\right)\frac{\pi}{2}$
x^2 / x^4	$\frac{2}{3}\sqrt{1-x^2}\left(1+2x^2\right)$	$\left(1-x^6\right)\frac{\pi}{3}$
x^3 / x^6	$\frac{1}{4}\left(\sqrt{1-x^2}\left(2+3x^2\right) + 3x^4 \log\left(\frac{1+\sqrt{1-x^2}}{x}\right)\right)$	$\left(1-x^8\right)\frac{\pi}{4}$
x^4 / x^8	$\frac{2}{5}\sqrt{1-x^2}\left(3+4x^2+8x^4\right)$	$\left(1-x^{10}\right)\frac{\pi}{5}$

Table D.1: Pre-integrated polynomials for the range $[0, 1]$.

filter (radius)	c_0	c_1	c_2	c_3	c_4	ε_{max}
Gaussian (2.3)	1.30321	0.119155	-9.69022	14.2894	-6.02528	0.00579162
Lanczos (2)	1.68971	0.71905	-19.8035	31.601	-14.2415	0.00892035
Mitchell-Netravali	2.03111	-2.16092	-15.102	31.24	-16.13328	0.0347118
B	-1.16265	2.328	8.46852	-21.7564	12.2412	0.0332397
C	-0.521127	3.9474	-4.20988	-4.72515	5.66	0.0445042
Blackman-Harris	1.36055	0.33004	-11.613	17.2908	-7.386368	0.00457847

Table D.2: Coefficients for polynomials fits of various 2D filters. All filters are normalized to the range $[0, 1]$.

filter (radius)	c_0	c_2	c_4	c_6	c_8	ε_{max}
Gaussian (2.3)	1.65951	-8.09529	16.6202	-16.1818	6.05909	0.00504176
Lanczos (2)	2.58949	-14.7969	30.5864	-27.9904	9.64687	0.00298654
Mitchell-Netravali	3.25801	-24.7342	65.8731	-73.6141	29.366	0.0131654
B	-2.16456	19.7691	-56.6125	65.4189	-26.5495	0.013814
C	-1.1118	16.1821	-56.1939	71.2608	-30.3158	0.0201162
Blackman-Harris	1.79798	-8.86188	17.5986	-16.2609	5.75158	0.00211148

Table D.3: Coefficients for polynomials fits of various 3D filters. All filters are normalized to the range $[0, 1]$.

List of Figures

List of Tables

List of Algorithms

Acronyms

AMD Advanced Micro Devices. 26

API Application Programming Interface. 8, 26, 83, 99

CPR Curved Planar Reformation (Kanitsar et al. 2002). 33–36, 38, 39, 113, 114

CPU Central Processing Unit. 24, 27, 29, 49, 83, 86

CR Centerline Reformation (Mistelbauer et al. 2012). 33–35, 38, 52–57, 113, 114

CREW Concurrent Read Exclusive Write. 25

CSAA Coverage Sampling Anti-Aliasing (Young 2006). 25, 102, 103

CSR Curved Surface Reformation. 34–36, 38, 46, 47, 49, 52–57, 113, 114

CTA Computed Tomography Angiography. 33, 53, 54, 56

CUDA Compute Unified Device Architecture (Nickolls et al. 2008). 8, 38, 49, 64, 65, 82, 83, 85, 86, 99

DOF Depth of Field. 22, 25

DSA Digital Subtraction Angiography. 33

DVR Direct Volume Rendering. 35–37, 54

FWHM full width at half maximum. 46

GIF Graphics Interchange Format. 3

GPGPU General-Purpose Computing on Graphics Processing Units. 26, 27, 66, 96, 99, 108

GPU Graphics Processing Unit. 7, 8, 13, 25–29, 38, 49, 64–66, 75, 82, 83, 85, 86, 89, 90, 96, 108

GVD Generalized Voronoi Diagram. 6, 107

JPEG Joint Photographics Experts Group. 3

LOD Level of Detail. 5, 24, 34, 43–46, 49, 50, 52, 54, 56, 88, 89, 107

MIDA Maximum Intensity Difference Accumulation (Bruckner and Gröller 2009). 49, 52–54

MIP Maximum Intensity Projection. 36, 37

MLAA Morphological Anti-Aliasing (Reshetov 2009). 25

mpCPR Multipath Curved Planar Reformation. 35, 36, 38, 39, 52–57, 113

MRA Magnetic Resonance Angiography. 33

MSAA Multisample Anti-Aliasing. 25, 94, 103

NSAA Non-Sampled Anti-Aliasing. 94, 95, 100, 101

OpenGL Open Graphics Library. 26, 49, 103

PE Processing Element. 27, 29

PNG Portable Network Graphics. 3

PRAM Parallel Random-Access Machine. 25

SIMD Single Instruction Multiple Data. 7, 8, 26–30, 65, 73, 76, 79, 82, 83, 85, 90

SMAA Enhanced Subpixel Morphological Anti-Aliasing (Jimenez et al. 2012). 103

SPMD Single Program Multiple Data. 27

SSAA Supersample Anti-Aliasing. 22, 94

Bibliography

Akeley, K. (1993). „Reality Engine graphics“. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '93. Anaheim, CA: ACM, pp. 109–116. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166131 (cit. on pp. 25–26).

Appel, A. (1967). „The notion of quantitative invisibility and the machine rendering of solids“. In: *Proceedings of the 1967 22nd National Conference*. ACM '67, pp. 387–393. DOI: 10.1145/800196.806007 (cit. on pp. 24, 76, 78).

Appel, A., F. J. Rohlf, and A. J. Stein (1979). „The haloed line effect for hidden line elimination“. In: *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '79, pp. 151–157. DOI: 10.1145/800249.807437 (cit. on p. 24).

Auzinger, T., M. Guthe, and S. Jeschke (2012). „Analytic Anti-Aliasing of Linear Functions on Polytopes“. In: *Computer Graphics Forum* 31.2pt1, pp. 335–344. ISSN: 0167-7055. DOI: 10.1111/j.1467-8659.2012.03012.x (cit. on pp. 7, 75).

Auzinger, T., G. Mistelbauer, I. Baclija, R. Schernthaner, A. Kochl, M. Wimmer, M. E. Groller, and S. Bruckner (2013). „Vessel Visualization using Curved Surface Reformation“. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12, pp. 2858–2867. ISSN: 1077-2626. DOI: 10.1109/tvcg.2013.215 (cit. on p. 6).

Auzinger, T., P. Musialski, R. Preiner, and M. Wimmer (2013). „Non-Sampled Anti-Aliasing“. In: *Vision, Modeling & Visualization*. Ed. by M. Bronstein, J. Favre, and K. Hormann. VMV '13. The Eurographics Association, pp. 169–176. DOI: 10.2312/PE.VMV.VMV13.169-176 (cit. on p. 7).

Auzinger, T., M. Wimmer, and S. Jeschke (2013). „Analytic Visibility on the GPU“. In: *Computer Graphics Forum* 32.2pt4, pp. 409–418. ISSN: 0167-7055. DOI: 10.1111/cgf.12061 (cit. on p. 7).

Balsa Rodríguez, M., E. Gobbetti, J. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. Suter (2014). „State-of-the-Art in Compressed GPU-Based Direct Volume Rendering“. In: *Computer Graphics Forum* 33.6, pp. 77–100. ISSN: 1467-8659. DOI: 10.1111/cgf.12280 (cit. on p. 27).

Barringer, R. and T. Akenine-Möller (2013). „A4: Asynchronous Adaptive Anti-aliasing Using Shared Memory“. In: *ACM Trans. Graph.* 32.4, 100:1–100:10. ISSN: 0730-0301. DOI: 10.1145/2461912.2462015 (cit. on p. 109).

Birkeland, Å., S. Bruckner, A. Brambilla, and I. Viola (2012). „Illustrative Membrane Clipping“. In: *Computer Graphics Forum* 31.3, pp. 905–914. DOI: 10.1111/j.1467-8659.2012.03083.x (cit. on p. 37).

Blinn, J. F. (1989). „Jim Blinn's corner - Return of the Jaggy (high frequency filtering)“. In: *IEEE Comput. Grap. Appl.* 9.2, pp. 82–89. ISSN: 0272-1716. DOI: 10.1109/38.19054 (cit. on p. 18).

Bracewell, R. N. (2000). *The Fourier Transform and Its Applications*. 3rd ed. Electrical engineering series. McGraw Hill. ISBN: 9780073039381. DOI: 10.1036/0073039381 (cit. on p. 17).

Bruckner, S. and M. E. Gröller (2009). „Instant Volume Visualization using Maximum Intensity Difference Accumulation“. In: *Proceedings of the 11th Eurographics / IEEE - VGTC conference on Visualization*. EuroVis '09. Eurographics Association, pp. 775–782. DOI: 10.1111/j.1467-8659.2009.01474.x (cit. on pp. 49, 144).

Buades, A., B. Coll, and J.-M. Morel (2005). „A Non-Local Algorithm for Image Denoising“. In: *Computer Vision and Pattern Recognition*. Vol. 2. CVPR '05. Institute of Electrical & Electronics Engineers (IEEE), pp. 60–65. ISBN: http://id.crossref.org/isbn/0-7695-2372-2. DOI: 10.1109/cvpr.2005.38 (cit. on p. 18).

Buck, I., T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan (2004). „Brook for GPUs: Stream Computing on Graphics Hardware“. In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. Los Angeles, California: ACM, pp. 777–786. DOI: 10.1145/1186562.1015800 (cit. on p. 26).

Burns, M. and A. Finkelstein (2008). „Adaptive cutaways for comprehensible rendering of polygonal scenes“. In: *ACM Trans. Graph.* 27.5, 154:1–154:7. DOI: 10.1145/1409060.1409107 (cit. on p. 36).

Burns, M., M. Haidacher, W. Wein, I. Viola, and M. E. Gröller (2007). „Feature Emphasis and Contextual Cutaways for Multimodal Medical Visualization“. In: *Proceedings of Eurographics / IEEE VGTC Symposium on Visualization*. Ed. by K. Museth, T. Moeller, and A. Ynnerman. EuroVis '07. Eurographics Association, pp. 275–282. ISBN: 978-3-905673-45-6. DOI: 10.2312/VisSym/EuroVis07/275-282 (cit. on p. 36).

Calatrava Moreno, M. d. C. and T. Auzinger (2013). „General-Purpose Graphics Processing Units in Service-Oriented Architectures“. In: *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*. SOCA '13. IEEE, pp. 260–267. ISBN: 978-1-4799-2701-2. DOI: 10.1109/soca.2013.15 (cit. on pp. 26–28).

Candes, E. J., J. Romberg, and T. Tao (2006). „Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information“. In: *IEEE Trans. Inform. Theory* 52.2, pp. 489–509. ISSN: 0018-9448. DOI: 10.1109/tit.2005.862083 (cit. on p. 16).

Catmull, E. (1978). „A hidden-surface algorithm with anti-aliasing“. In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '78. New York, NY, USA: ACM, pp. 6–11. DOI: 10.1145/800248.807360 (cit. on pp. 23–24).

— (1984). „An analytic visible surface algorithm for independent pixel processing“. In: *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '84. New York, NY, USA: ACM, pp. 109–115. ISBN: 0-89791-138-5. DOI: 10.1145/800031.808586 (cit. on pp. 23–24).

Chajdas, M. G., M. McGuire, and D. Luebke (2011). „Subpixel reconstruction antialiasing for deferred shading“. In: *Proceedings of the 2011 Symposium on Interactive 3D Graphics and Games*. I3D '11. San Francisco, California: ACM, pp. 15–22. ISBN: 978-1-4503-0565-5. DOI: 10.1145/1944745.1944748 (cit. on p. 25).

Chang, P. and R. Jain (1981). „A multi-processor system for hidden-surface-removal“. In: *SIGGRAPH Computer Graphics* 15.4, pp. 405–436. doi: 10.1145/988428.988434 (cit. on p. 24).

Chuon, C., S. Guha, P. Janecek, and N. D. C. Song (2011). „Simplipoly: Curvature-Based Polygonal Curve Simplification“. In: *International Journal of Computational Geometry & Applications* 21.4, pp. 417–429. doi: 10.1142/S0218195911003743 (cit. on pp. 37, 45).

Clarberg, P. and J. Munkberg (2014). „Deep shading buffers on commodity GPUs“. In: *ACM Transactions on Graphics* 33.6, pp. 1–12. issn: 0730-0301. doi: 10.1145/2661229.2661245 (cit. on p. 25).

Clarberg, P., R. Toth, J. Hasselgren, J. Nilsson, and T. Akenine-Möller (2014). „AMFS: Adaptive Multi-Frequency Shading for Future Graphics Processors“. In: *ACM Transactions on Graphics* 33.4, pp. 1–12. issn: 0730-0301. doi: 10.1145/2601097.2601214 (cit. on p. 25).

Correa, C., P. Lindstrom, and P.-T. Bremer (2011). „Topological Spines: A Structure-preserving Visual Representation of Scalar Fields“. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12, pp. 1842–1851. doi: 10.1109/TVCG.2011.244 (cit. on p. 57).

Crow, F. C. (1977). „The aliasing problem in computer-generated shaded images“. In: *Commun. ACM* 20 (11), pp. 799–805. issn: 0001-0782. doi: 10.1145/359863.359869 (cit. on pp. 23, 94).

Dachille, F. and A. E. Kaufman (2000). „Incremental Triangle Voxelization“. In: *Proceedings of the Graphics Interface 2000 Conference*. Ed. by S. Fels and P. Poulin. GI '00. Canadian Human-Computer Communications Society, pp. 205–212. isbn: 0-9695338-9-6 (cit. on p. 69).

Darema, F., D. George, V. Norton, and G. Pfister (1988). „A single-program-multiple-data computational model for EPEX/FORTRAN“. In: *Parallel Computing* 7.1, pp. 11–24. issn: 0167-8191. doi: 10.1016/0167-8191(88)90094-4 (cit. on p. 27).

DeCarlo, D., A. Finkelstein, S. Rusinkiewicz, and A. Santella (2003). „Suggestive contours for conveying shape“. In: *ACM Transactions on Graphics* 22.3, pp. 848–855. doi: 10.1145/1201775.882354 (cit. on p. 36).

Deering, M., S. Winner, B. Schediwy, C. Duffy, and N. Hunt (1988). „The triangle processor and normal vector shader: a VLSI system for high performance graphics“. In: *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '88. New York, NY, USA: ACM, pp. 21–30. isbn: 0-89791-275-6. doi: 10.1145/54852.378468 (cit. on p. 25).

Dévai, F. (2011). „An optimal hidden-surface algorithm and its parallelization“. In: *Proceedings of the 2011 International Conference on Computational Science and its Applications*. ICCSA '11. Springer, pp. 17–29. doi: 10.1007/978-3-642-21931-3_2 (cit. on pp. 24, 73, 75, 78).

Diepstraten, J., D. Weiskopf, and T. Ertl (2003). „Interactive Cutaway Illustrations“. In: *Computer Graphics Forum* 22.3, pp. 523–532. doi: 10.1111/1467-8659.t01-3-00700 (cit. on p. 36).

Douglas, D. H. and T. K. Peucker (1973). „Algorithms for the reduction of the number of points required to represent a digitized line or its caricature“. In: *Cartographica* 10.2, pp. 112–122. doi: 10.3138/FM57-6770-U75U-7727 (cit. on p. 37).

Duchon, C. E. (1979). „Lanczos Filtering in One and Two Dimensions“. In: *Journal of Applied Meteorology* 18 (8), pp. 1016–1022 (cit. on p. 17).

Duff, T. (1989). „Polygon scan conversion by exact convolution“. In: *Raster Imaging and Digital Typography*. Ed. by J. André and R. D. Hersch. Vol. 1. Cambridge University Press, pp. 154–168 (cit. on pp. 23, 59).

Durand, F. (2000). „A Multidisciplinary Survey of Visibility“. In: *ACM SIGGRAPH Courses*. SIGGRAPH '00. ACM (cit. on p. 24).

Ebeida, M. S., A. Patney, S. A. Mitchell, K. R. Dalbey, A. A. Davidson, and J. D. Owens (2014). „k-d Darts: Sampling by k-dimensional flat searches“. In: *ACM Transactions on Graphics* 33.1, pp. 1–16. ISSN: 0730-0301. DOI: 10.1145/2522528 (cit. on p. 25).

Elber, G. and E. Cohen (1990). „Hidden curve removal for free form surfaces“. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '90. Association for Computing Machinery (ACM), pp. 95–104. ISBN: 0201509334. DOI: 10.1145/97879.97890 (cit. on p. 24).

Everts, M. H., H. Bekker, J. B. Roerdink, and T. Isenberg (2009). „Depth-Dependent Halos: Illustrative Rendering of Dense Line Data“. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6, pp. 1299–1306. ISSN: 1077-2626. DOI: 10.1109/TVCG.2009.138 (cit. on p. 41).

Feibush, E. A., M. Levoy, and R. L. Cook (1980). „Synthetic texturing using digital filters“. In: *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '80. Seattle, Washington, United States: ACM, pp. 294–301. ISBN: 0-89791-021-4. DOI: 10.1145/800250.807507 (cit. on p. 23).

Flynn, M. J. (1972). „Some Computer Organizations and Their Effectiveness“. In: *IEEE Trans. Comput.* 21.9, pp. 948–960. ISSN: 0018-9340. DOI: 10.1109/TC.1972.5009071 (cit. on p. 27).

Frankel, A., D. Nussbaum, and J.-R. Sack (2004). „Floating-Point Filter for the Line Intersection Algorithm“. In: *Geographic Information Science*. Ed. by M. J. Egenhofer, C. Freksa, and H. J. Miller. Vol. 3234. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 94–105. ISBN: 978-3-540-30231-5. DOI: 10.1007/978-3-540-30231-5_7 (cit. on p. 77).

Franklin, W. R. (1980). „A linear time exact hidden surface algorithm“. In: *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '80, pp. 117–123. DOI: 10.1145/800250.807480 (cit. on p. 24).

Galimberti, R. (1969). „An algorithm for hidden line elimination“. In: *Communications of the ACM* 12.4, pp. 206–211. DOI: 10.1145/362912.362921 (cit. on p. 24).

Ganacim, F., R. S. Lima, L. H. de Figueiredo, and D. Nehab (2014). „Massively-parallel vector graphics“. In: *ACM Transactions on Graphics* 33.6, pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/2661229.2661274 (cit. on p. 25).

Grant, C. W. (1985). „Integrated analytic spatial and temporal anti-aliasing for polyhedra in 4-space“. In: *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '85. New York, NY, USA: ACM, pp. 79–84. ISBN: 0-89791-166-0. DOI: 10.1145/325334.325184 (cit. on p. 23).

Gribel, C. J., R. Barringer, and T. Akenine-Möller (2011). „High-quality Spatio-temporal Rendering Using Semi-analytical Visibility“. In: *ACM Trans. Graph.* 30.4, 54:1–54:12. ISSN: 0730-0301. DOI: 10.1145/2010324.1964949 (cit. on p. 25).

Gribel, C. J., M. Doggett, and T. Akenine-Möller (2010). „Analytical Motion Blur Rasterization with Compression“. In: *Proceedings of the Conference on High Performance Graphics*. Ed. by M. Doggett, S. Laine, and W. Hunt. HPG '10. Eurographics Association, pp. 163–172. isbn: 978-3-905674-26-2. doi: 10.2312/EGGH/HPG10/163-172 (cit. on p. 25).

Gudmundsson, J., G. Narasimhan, and M. Smid (2007). „Distance-preserving approximations of polygonal paths“. In: *Computational Geometry* 36.3, pp. 183–196. doi: 10.1016/j.comgeo.2006.05.002 (cit. on p. 37).

Guenter, B. and J. Tumblin (1996). „Quadrature prefiltering for high quality antialiasing“. In: *ACM Trans. Graph.* 15 (4), pp. 332–353. issn: 0730-0301. doi: 10.1145/234535.234540 (cit. on pp. 23, 62).

Gupta, N. and S. Sen (1998). „An improved output-size sensitive parallel algorithm for hidden-surface removal for terrains“. In: *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*. IPPS/SPDP '98, pp. 215–219. doi: 10.1109/IPPS.1998.669913 (cit. on p. 24).

Gupta, S., R. F. Sproull, and I. E. Sutherland (1981). „A VLSI Architecture for Updating Raster-scan Displays“. In: *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '81. Dallas, Texas, USA: ACM, pp. 71–78. isbn: 0-89791-045-1. doi: 10.1145/800224.806791 (cit. on p. 26).

Hasselgren, J., T. Akenine-Möller, and L. Ohlsson (2005). „Conservative Rasterization“. In: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Ed. by M. Pharr and R. Fernando. 1st ed. Vol. 2. GPU Gems. Addison-Wesley Professional. Chap. 42, pp. 677–690. isbn: 0321335597 (cit. on pp. 23, 97).

He, Y., Y. Gu, and K. Fatahalian (2014). „Extending the graphics pipeline with adaptive, multi-rate shading“. In: *ACM Transactions on Graphics* 33.4, pp. 1–12. issn: 0730-0301. doi: 10.1145/2601097.2601105 (cit. on p. 25).

Hoberock, J. and N. Bell (2010). *Thrust: A Parallel Template Library*. Version 1.8.0 (cit. on pp. 84, 96).

Hoff III, K. E., J. Keyser, M. Lin, D. Manocha, and T. Culver (1999). „Fast computation of generalized Voronoi diagrams using graphics hardware“. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co., pp. 277–286. doi: 10.1145/311535.311567 (cit. on pp. 6, 52).

Hornung, C. (1982). „An approach to a calculation-minimized hidden line algorithm“. In: *Computers & Graphics* 6.3, pp. 121–126. doi: 10.1016/0097-8493(82)90005-X (cit. on p. 24).

Huang, J., R. Yagel, V. Filippov, and Y. Kurzion (1998). „An accurate method for voxelizing polygon meshes“. In: *Proceedings of the 1998 IEEE symposium on Volume visualization*. VVS '98. Research Triangle Park, North Carolina, United States: ACM, pp. 119–126. isbn: 1-58113-105-4. doi: 10.1145/288126.288181 (cit. on p. 23).

Imai, H. and M. Iri (1986). „Computational-geometric methods for polygonal approximations of a curve“. In: *Computer Vision, Graphics, and Image Processing* 36.1, pp. 31–41. doi: 10.1016/S0734-189X(86)80027-5 (cit. on p. 37).

Imai, H. and M. Iri (1988). „Computational Morphology“. In: North-Holland. Chap. Polygonal Approximations of a Curve – Formulations and Algorithms, pp. 87–95 (cit. on p. 37).

James, J. (1995). *A student's guide to Fourier transforms: with applications in physics and engineering*. New York: Cambridge University Press. isbn: 9780521468299 (cit. on p. 19).

Jankun-Kelly, M., M. Jiang, D. Thompson, and R. Machiraju (2006). „Vortex Visualization for Practical Engineering Applications“. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5, pp. 957–964. doi: 10.1109/TVCG.2006.201 (cit. on p. 57).

Jiang, X.-d., B. Sheng, W.-y. Lin, W. Lu, and L.-z. Ma (2014). „Image anti-aliasing techniques for Internet visual media processing: a review“. In: *Journal of Zhejiang University SCIENCE C* 15.9, pp. 717–728. issn: 1869-1951. doi: 10.1631/jzus.C1400100 (cit. on p. 23).

Jianu, R., C. Demiralp, and D. H. Laidlaw (2012). „Exploring Brain Connectivity with Two-Dimensional Neural Maps“. In: *IEEE Transactions on Visualization and Computer Graphics* 18.6, pp. 978–987. issn: 1077-2626. doi: 10.1109/TVCG.2011.82 (cit. on p. 35).

Jimenez, J., J. I. Echevarria, T. Sousa, and D. Gutierrez (2012). „SMAA: Enhanced Subpixel Morphological Antialiasing“. In: *Computer Graphics Forum* 31.2pt1, pp. 355–364. issn: 0167-7055. doi: 10.1111/j.1467-8659.2012.03014.x (cit. on pp. 25, 103, 144).

Jimenez, J., D. Gutierrez, J. Yang, A. Reshetov, P. Demoreuille, T. Berghoff, C. Perthuis, H. Yu, M. McGuire, T. Lottes, H. Malan, E. Persson, D. Andreev, and T. Sousa (2011). „Filtering approaches for real-time anti-aliasing“. In: *ACM SIGGRAPH 2011 Courses*. SIGGRAPH '11. Vancouver, British Columbia, Canada: ACM, 6:1–6:329. isbn: 978-1-4503-0967-7. doi: 10.1145/2037636.2037642 (cit. on p. 26).

Jones, M. W. (1996). „The production of volume data from triangular meshes using voxelisation“. In: *Computer Graphics Forum* 15.5, pp. 311–318. doi: 10.1111/1467-8659.1550311 (cit. on p. 23).

Kajiya, J. and M. Ullner (1981). „Filtering high quality text for display on raster scan devices“. In: *ACM Trans. Graph.* SIGGRAPH '81. ACM, pp. 7–15. isbn: 0-89791-045-1. doi: 10.1145/965161.806784 (cit. on p. 23).

Kanitsar, A., D. Fleischmann, R. Wegenkittl, P. Felkel, and M. E. Gröller (2002). „CPR - Curved Planar Reformation“. In: *Proceedings of IEEE Visualization*. VIS '02. IEEE, pp. 37–44. doi: 10.1109/VISUAL.2002.1183754 (cit. on pp. 34, 143).

Kanitsar, A., D. Fleischmann, R. Wegenkittl, and M. E. Gröller (2006). „Diagnostic Relevant Visualization of Vascular Structures“. In: *Scientific Visualization: The Visual Extraction of Knowledge from Data*. Ed. by G.-P. Bonneau, T. Ertl, and G. Nielson. Mathematics and Visualization. Springer Berlin Heidelberg, pp. 207–228. isbn: 978-3-540-30790-7. doi: 10.1007/3-540-30790-7_13 (cit. on pp. 35, 38).

Kanitsar, A., D. Fleischmann, R. Wegenkittl, D. Sandner, P. Felkel, and M. E. Gröller (2001). „Computed tomography angiography: a case study of peripheral vessel investigation“. In: *Proceedings of IEEE Visualization*. VIS '01. IEEE, pp. 477–593. doi: 10.1109/VISUAL.2001.964555 (cit. on p. 37).

Kanitsar, A., R. Wegenkittl, D. Fleischmann, and E. Gröller (2003). „Advanced Curved Planar Reformation: Flattening of Vascular Structures“. In: *Proceedings of IEEE Visualization*. VIS '03. IEEE, pp. 43–50. doi: 10.1109/VISUAL.2003.1250353 (cit. on p. 35).

Karras, T. and T. Aila (2013). „Fast Parallel Construction of High-quality Bounding Volume Hierarchies“. In: *Proceedings of the 5th High-Performance Graphics Conference*. HPG '13. Anaheim, California: ACM, pp. 89–99. isbn: 978-1-4503-2135-8. doi: 10.1145/2492045.2492055 (cit. on p. 27).

Kaufman, A. E. (1987). „Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes“. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '87. New York, NY, USA: ACM, pp. 171–179. isbn: 0-89791-227-6. doi: 10.1145/37401.37423 (cit. on p. 23).

Kenwright, D. N. and R. Haimes (1998). „Automatic Vortex Core Detection“. In: *IEEE Computer Graphics and Applications* 18.4, pp. 70–74. doi: 10.1109/38.689668 (cit. on p. 57).

Konrad-Verse, O., B. Preim, and A. Littmann (2004). „Virtual Resection with a Deformable Cutting Plane“. In: *Proceedings of Simulation und Visualisierung 2004*, pp. 203–214 (cit. on p. 37).

Křivánek, J., A. Keller, I. Georgiev, A. S. Kaplanyan, M. Fajardo, M. Meyer, J.-D. Nahmias, O. Karlík, and J. Cañada (2014). „Recent advances in light transport simulation“. In: *ACM SIGGRAPH 2014 Courses*. SIGGRAPH '14. ACM, Article No. 17. isbn: 9781450329620. doi: 10.1145/2614028.2615438 (cit. on p. 22).

Krüger, J. and R. Westermann (2003). „Linear Algebra Operators for GPU Implementation of Numerical Algorithms“. In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. San Diego, California: ACM, pp. 908–916. isbn: 1-58113-709-5. doi: 10.1145/1201775.882363 (cit. on p. 26).

Laine, S. and T. Karras (2011). „High-performance Software Rasterization on GPUs“. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. Vancouver, British Columbia, Canada: ACM, pp. 79–88. isbn: 978-1-4503-0896-0. doi: 10.1145/2018323.2018337 (cit. on p. 26).

Lampe, O. D., C. Correa, K.-L. Ma, and H. Hauser (2009). „Curve-Centric Volume Reformation for Comparative Visualization“. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6, pp. 1235–1242. doi: 10.1109/TVCG.2009.136 (cit. on p. 35).

Lee, N. and M. Rasch (2006). „Tangential curved planar reformation for topological and orientation invariant visualization of vascular trees“. In: *Engineering in Medicine and Biology Society, 28th Annual International Conference of the IEEE on*. EMBS '06. IEEE, pp. 1073–1076. doi: 10.1109/IEMBS.2006.259518 (cit. on p. 35).

Levinthal, A., P. Hanrahan, M. Paquette, and J. Lawson (1987). „Parallel Computers for Graphics Applications“. In: *Proceedings of the Second International Conference on Architectual Support for Programming Languages and Operating Systems*. ASPLOS II. Palo Alto, California, USA: IEEE Computer Society Press, pp. 193–198. isbn: 0-8186-0805-6. doi: 10.1145/36206.36202 (cit. on p. 26).

Levinthal, A. and T. Porter (1984). „Chap - a SIMD Graphics Processor“. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. New York, NY, USA: ACM, pp. 77–82. isbn: 0-89791-138-5. doi: 10.1145/800031.808581 (cit. on p. 26).

Li, W., L. Ritter, M. Agrawala, B. Curless, and D. Salesin (2007). „Interactive cutaway illustrations of complex 3D models“. In: *ACM Trans. Graph.* 26.3, 31:1–31:11. doi: 10.1145/1276377.1276416 (cit. on p. 36).

Lidal, E. M., H. Hauser, and I. Viola (2012). „Design principles for cutaway visualization of geological models“. In: *Proceedings of the 28th Spring Conference on Computer Graphics*. SCCG '12. ACM, pp. 47–54. doi: 10.1145/2448531.2448537 (cit. on p. 36).

Liktor, G. and C. Dachsbacher (2012). „Decoupled deferred shading for hardware rasterization“. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '12. ACM, pp. 143–150. isbn: 9781450311946. doi: 10.1145/2159616.2159640 (cit. on p. 25).

Lin, Z., H.-T. Chen, H.-Y. Shum, and J. Wang (2005). „Prefiltering Two-Dimensional Polygons without Clipping“. In: *Journal of Graphics, GPU, and Game Tools* 10.1, pp. 17–26. issn: 2151-2272. doi: 10.1080/2151237x.2005.10129189 (cit. on p. 23).

Lindholm, E., M. J. Kilgard, and H. Moreton (2001). „A User-programmable Vertex Engine“. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, pp. 149–158. isbn: 1-58113-374-X. doi: 10.1145/383259.383274 (cit. on p. 26).

Lottes, T. (2009). *FXAA-Whitepaper*. Tech. rep. NVIDIA Corporation (cit. on p. 25).

Manson, J. and S. Schaefer (2011). „Wavelet Rasterization“. In: *Computer Graphics Forum* 30.2, pp. 395–404. issn: 0167-7055. doi: 10.1111/j.1467-8659.2011.01887.x (cit. on pp. 24, 75).

— (2013). „Analytic Rasterization of Curves with Polynomial Filters“. In: *Computer Graphics Forum* 32.2pt4, pp. 499–507. issn: 0167-7055. doi: 10.1111/cgf.12070 (cit. on pp. 24, 98, 109).

Mark, W. R., R. S. Glanville, K. Akeley, and M. J. Kilgard (2003). „Cg: A System for Programming Graphics Hardware in a C-like Language“. In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. San Diego, California: ACM, pp. 896–907. isbn: 1-58113-709-5. doi: 10.1145/1201775.882362 (cit. on p. 26).

McCool, M. D. (1995). „Analytic antialiasing with prism splines“. In: *ACM Trans. Graph.* SIGGRAPH '95. ACM, pp. 429–436. isbn: 0-89791-701-4. doi: 10.1145/218380.218499 (cit. on p. 23).

McGuffin, M. J., L. Tancau, and R. Balakrishnan (2003). „Using Deformations for Browsing Volumetric Data“. In: *Proceedings of IEEE Visualization*. VIS '03. IEEE, pp. 401–408. doi: 10.1109/VISUAL.2003.1250400 (cit. on p. 36).

McGuire, M., E. Enderton, P. Shirley, and D. Luebke (2010). „Real-time Stochastic Rasterization on Conventional GPU Architectures“. In: *Proceedings of the Conference on High Performance Graphics*. Ed. by M. Doggett, S. Laine, and W. Hunt. HPG '10. Eurographics Association, pp. 173–182. isbn: 978-3-905674-26-2. doi: 10.2312/EGGH/HPG10/173-182 (cit. on p. 25).

McGuire, M., P. Hennessy, M. Bukowski, and B. Osman (2012). „A reconstruction filter for plausible motion blur“. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '12. ACM, pp. 135–142. isbn: 9781450311946. doi: 10.1145/2159616.2159639 (cit. on p. 25).

McInerney, T. and P. Crawford (2010). „RibbonView: interactive context-preserving cutaways of anatomical surface meshes“. In: *Proceedings of the 6th International Symposium on Advances in Visual Computing*. Ed. by G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. Chung, R. Hammound, M. Hussain, T. Kar-Han, R. Crawfis, D. Thalmann, D. Kao, and L. Avila. Vol. 2. ISVC '10, pp. 533–544. isbn: 78-3-642-17274-8. doi: 10.1007/978-3-642-17274-8_52 (cit. on p. 36).

McKenna, M. (1987). „Worst-case optimal hidden-surface removal“. In: *ACM Trans. Graph.* 6.1, pp. 19–28. doi: 10.1145/27625.27627 (cit. on p. 24).

Merrill, D. and A. Grimshaw (2011). „High Performance and Scalable Radix Sorting: A case study of implementing dynamic parallelism for GPU computing“. In: *Parallel Processing Letters* 21.02, pp. 245–272. doi: 10.1142/S0129626411000187 (cit. on p. 84).

Mistelbauer, G. (2013). „Smart Interactive Vessel Visualization in Radiology“. PhD thesis. Vienna University of Technology (cit. on p. 34).

Mistelbauer, G., A. Morar, A. Varchola, R. Schernthaner, I. Baclija, A. Köchl, A. Kanitsar, S. Bruckner, and M. E. Gröller (2013). „Vessel Visualization using Curvicircular Feature Aggregation“. In: *Computer Graphics Forum* 32.3, pp. 231–240. doi: 10.1111/cgf.12110 (cit. on pp. 35, 57).

Mistelbauer, G., A. Varchola, H. Bouzari, J. Starinsky, A. Köchl, R. Schernthaner, D. Fleischmann, M. E. Gröller, and M. Srámek (2012). „Centerline reformations of complex vascular structures“. In: *Pacific Visualization Symposium, 2012 IEEE*. PacificVis '12. IEEE, pp. 233–240. doi: 10.1109/PacificVis.2012.6183596 (cit. on pp. 35, 38, 143).

Mitchell, D. P. and A. N. Netravali (1988). „Reconstruction filters in computer-graphics“. In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '88. ACM, pp. 221–228. isbn: 0-89791-275-6. doi: 10.1145/54852.378514 (cit. on pp. 18, 20, 23).

Mulmuley, K. (1989). „An efficient algorithm for hidden surface removal“. In: *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '89. ACM, pp. 379–388. doi: 10.1145/74333.74372 (cit. on p. 24).

Nealen, A., M. Müller, R. Keiser, E. Boxerman, and M. Carlson (2006). „Physically Based Deformable Models in Computer Graphics“. In: *Computer Graphics Forum* 25.4, pp. 809–836. doi: 10.1111/j.1467-8659.2006.01000.x (cit. on p. 37).

Nickolls, J., I. Buck, M. Garland, and K. Skadron (2008). „Scalable parallel programming with CUDA“. In: *Queue* 6.2, p. 40. issn: 1542-7730. doi: 10.1145/1365490.1365500 (cit. on p. 143).

Nyquist, H. (2002). „Certain topics in telegraph transmission theory“. In: *Proceedings of the IEEE* 90.2, pp. 280–305. issn: 0018-9219. doi: 10.1109/5.989875 (cit. on p. 16).

Pantaleoni, J. (2011a). „VoxelPipe: A Programmable Pipeline for 3D Voxelization“. In: *Proceedings of the 2011 ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. ACM, pp. 99–106. isbn: 9781450308960. doi: 10.1145/2018323.2018339 (cit. on pp. 23, 84).

— (2011b). „VoxelPipe: A Programmable Pipeline for 3D Voxelization“. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. Vancouver, British Columbia, Canada: ACM, pp. 99–106. isbn: 978-1-4503-0896-0. doi: 10.1145/2018323.2018339 (cit. on p. 27).

Parker, S. G., H. Friedrich, D. Luebke, K. Morley, J. Bigler, J. Hoberock, D. McAllister, A. Robison, A. Dietrich, G. Humphreys, M. McGuire, and M. Stich (2013). „GPU Ray Tracing“. In: *Commun. ACM* 56.5, pp. 93–101. ISSN: 0001-0782. DOI: 10.1145/2447976.2447997 (cit. on p. 26).

Petersen, D. P. and D. Middleton (1962). „Sampling and reconstruction of wave-number-limited functions in N-dimensional euclidean spaces“. In: *Information and Control* 5.4, pp. 279–323. ISSN: 0019-9958. DOI: 10.1016/s0019-9958(62)90633-2 (cit. on p. 20).

Powell, D. and T. Abel (2014). „An exact general remeshing scheme applied to conservative voxelization“. In: *ArXiv e-prints*. arXiv: 1412.4941 (cit. on p. 24).

Ragan-Kelley, J., J. Lehtinen, J. Chen, M. Doggett, and F. Durand (2011). „Decoupled sampling for graphics pipelines“. In: *ACM Trans. Graph.* 30.3, 17:1–17:17. ISSN: 0730-0301. DOI: 10.1145/1966394.1966396 (cit. on p. 25).

Ramer, U. (1972). „An iterative procedure for the polygonal approximation of plane curves“. In: *Computer Graphics and Image Processing* 1.3, pp. 244–256. DOI: 10.1016/S0146-664X(72)80017-0 (cit. on p. 37).

Randolph Franklin, W. and M. S. Kankanhalli (1990). „Parallel object-space hidden surface removal“. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '90. ACM, pp. 87–94. DOI: 10.1145/97879.97889 (cit. on p. 24).

Reif, J. H. and S. Sen (1988). „An efficient output-sensitive hidden surface removal algorithm and its parallelization“. In: *Proceedings of the fourth Annual Symposium on Computational Geometry*. SCG '88, pp. 193–200. DOI: 10.1145/73393.73413 (cit. on p. 24).

Reshetov, A. (2009). „Morphological antialiasing“. In: *Proceedings of the Conference on High Performance Graphics*. HPG '09. New Orleans, Louisiana: ACM, pp. 109–116. ISBN: 978-1-60558-603-8. DOI: 10.1145/1572769.1572787 (cit. on pp. 25, 144).

Roberts, L. G. (1963). „Machine Perception of Three-Dimensional Solids“. PhD thesis. Massachusetts Institute of Technology (cit. on p. 24).

Roos, J. E., D. Fleischmann, A. Köchl, T. Rakshe, M. Straka, A. Napoli, A. Kanitsar, M. Sramek, and E. Gröller (2007). „Multi-path Curved Planar Reformation (mpCPR) of the Peripheral Arterial Tree in CT Angiography (CTA)“. In: *Radiology* 244.1, pp. 281–290. DOI: 10.1148/radiol.2441060976 (cit. on p. 35).

Rusinkiewicz, S., F. Cole, D. DeCarlo, and A. Finkelstein (2008). „Line drawings from 3D models“. In: *ACM SIGGRAPH 2008 classes*. SIGGRAPH '08. ACM, 39:1–39:356. DOI: 10.1145/1401132.1401188 (cit. on p. 24).

Sander, P. V., X. Gu, S. J. Gortler, H. Hoppe, and J. Snyder (2000). „Silhouette clipping“. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. ACM Press, pp. 327–334. DOI: 10.1145/344779.344935 (cit. on p. 36).

Saroul, L., O. Figueiredo, and R.-D. Hersch (2006). „Distance Preserving Flattening of Surface Sections“. In: *IEEE Transactions on Visualization and Computer Graphics* 12.1, pp. 26–35. DOI: 10.1109/TVCG.2006.7 (cit. on p. 36).

Saroul, L., S. Gerlach, and R. D. Hersch (2003). „Exploring Curved Anatomic Structures with Surface Sections“. In: *Proceedings of IEEE Visualization*. VIS '03. IEEE, pp. 27–34. DOI: 10.1109/VISUAL.2003.1250351 (cit. on p. 36).

Satish, N., M. Harris, and M. Garland (2009). „Designing Efficient Sorting Algorithms for Manycore GPUs“. In: *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. IPDPS '09. Washington, DC, USA: IEEE Computer Society, pp. 1–10. isbn: 978-1-4244-3751-1. doi: 10.1109/IPDPS.2009.5161005 (cit. on p. 26).

Satish, N., C. Kim, J. Chhugani, H. Saito, R. Krishnaiyer, M. Smelyanskiy, M. Girkar, and P. Dubey (2012). „Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications?“ In: *Proceedings of the 39th Annual International Symposium on Computer Architecture*. ISCA '12. Portland, Oregon: IEEE Computer Society, pp. 440–451. isbn: 978-1-4503-1642-2. doi: 10.1145/2366231.2337210 (cit. on p. 27).

Schreiber, W. F. and D. E. Troxel (1985). „Transformation Between Continuous and Discrete Representations of Images: A Perceptual Approach“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 7.2 (2), pp. 178–186. doi: 10.1109/TPAMI.1985.4767642 (cit. on p. 23).

Schwartz, L. (1951–1957). *Théorie des distributions*. Vol. 1–2. Paris: Hermann (cit. on p. 14).

Sen, P. and S. Darabi (2010). „Compressive estimation for signal integration in rendering“. In: *Computer Graphics Forum* 29.4, pp. 1355–1363. issn: 1467-8659. doi: 10.1111/j.1467-8659.2010.01731.x (cit. on p. 16).

Sengupta, S., M. Harris, Y. Zhang, and J. D. Owens (2007). „Scan Primitives for GPU Computing“. In: *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*. Ed. by M. Segal and T. Aila. GH '07. San Diego, California: Eurographics Association, pp. 97–106. isbn: 978-1-59593-625-7. doi: 10.2312/EGGH/EGGH07/097-106 (cit. on p. 26).

Shannon, C. E. (1998). „Communication In The Presence Of Noise“. In: *Proceedings of the IEEE* 86.2, pp. 447–457. issn: 1558-2256. doi: 10.1109/jproc.1998.659497 (cit. on p. 16).

Sharir, M. and M. H. Overmars (1992). „A simple output-sensitive algorithm for hidden surface removal“. In: *ACM Trans. Graph.* 11.1, pp. 1–11. doi: 10.1145/102377.112141 (cit. on p. 24).

Sigg, S., R. Fuchs, R. Carnecky, and R. Peikert (2012). „Intelligent cutaway illustrations“. In: *Pacific Visualization Symposium (PacificVis), 2012 IEEE*. IEEE, pp. 185–192. doi: 10.1109/PacificVis.2012.6183590 (cit. on p. 36).

Sobolev, S. L. (1936). „Méthode nouvelle à résoudre le problème de Cauchy pour les équations linéaires hyperboliques normales.“ In: *Matematiceskij sbornik* 43.1, pp. 39–72 (cit. on p. 14).

Sramek, M. and A. E. Kaufman (1998). „Object voxelization by filtering“. In: *IEEE Symposium on Volume Visualization*. doi: 10.1109/svv.1998.729592 (cit. on p. 23).

— (1999). „Alias-Free Voxelization of Geometric Objects“. In: *IEEE Transactions on Visualization and Computer Graphics* 5 (3), pp. 251–267. issn: 1077-2626. doi: 10.1109/2945.795216 (cit. on p. 23).

Straka, M., A. Köchl, M. Cervenansky, M. Sramek, D. Fleischmann, A. L. Cruz, and E. Gröller (2004). „The VesselGlyph: Focus & Context Visualization in CT-Angiography“. In: *Proceedings of IEEE Visualization*. VIS '04. IEEE, pp. 385–392. doi: 10.1109/VISUAL.2004.104 (cit. on pp. 36, 49).

Strichartz, R. S. (2003). *A Guide to Distribution Theory and Fourier Transforms*. World Scientific Publishing Company. doi: 10.1142/5314 (cit. on p. 14).

Sun, X., K. Zhou, J. Guo, G. Xie, J. Pan, W. Wang, and B. Guo (2013). „Line segment sampling with blue-noise properties“. In: *ACM Transactions on Graphics* 32.4, Article No. 127. ISSN: 0730-0301. DOI: 10.1145/2461912.2462023 (cit. on p. 25).

Sutherland, I. E. (1964). „Sketchpad: A Man-Machine Graphical Communication System“. In: *Proceedings of the SHARE Design Automation Workshop*. DAC '64. New York, NY, USA: ACM, pp. 6.329–6.346. DOI: 10.1145/800265.810742 (cit. on pp. 26, 73).

Sutherland, I. E., R. F. Sproull, and R. A. Schumacker (1973). „Sorting and the hidden-surface problem“. In: *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*. AFIPS '73, pp. 685–693. DOI: 10.1145/1499586.1499749 (cit. on pp. 24, 75).

— (1974). „A Characterization of Ten Hidden-Surface Algorithms“. In: *ACM Computing Surveys* 6.1, pp. 1–55. DOI: 10.1145/356625.356626 (cit. on p. 24).

Tatarchuk, N., B. Karis, M. Drobot, N. Schulz, J. Charles, and T. Mader (2014). „Advances in Real-time Rendering in Games“. In: *ACM SIGGRAPH 2014 Courses*. SIGGRAPH '14. Vancouver, Canada: ACM, 10:1–10:1. ISBN: 978-1-4503-2962-0. DOI: 10.1145/2614028.2615455 (cit. on p. 26).

Tzeng, S., A. Patney, A. Davidson, M. S. Ebeida, S. A. Mitchell, and J. D. Owens (2012). „High-Quality Parallel Depth-of-Field Using Line Samples“. In: *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. Ed. by C. Dachsbacher, J. Munkberg, and J. Pantaleoni. HPG '12. Eurographics Association, pp. 23–31. ISBN: 978-3-905674-41-5. DOI: 10.2312/EGGH/HPG12/023-031 (cit. on p. 25).

Viola, I., M. Feixas, M. Sbert, and M. E. Gröller (2006). „Importance-Driven Focus of Attention“. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5, pp. 933–940. DOI: 10.1109/TVCG.2006.152 (cit. on p. 36).

Volkov, V. and J. W. Demmel (2008). „Benchmarking GPUs to Tune Dense Linear Algebra“. In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. SC '08. Austin, Texas: IEEE Press, 31:1–31:11. ISBN: 978-1-4244-2835-9. DOI: 10.1109/SC.2008.5214359 (cit. on p. 26).

Wald, I. (2012). „Fast Construction of SAH BVHs on the Intel Many Integrated Core (MIC) Architecture“. In: *IEEE Transactions on Visualization and Computer Graphics* 18.1, pp. 47–57. ISSN: 1077-2626. DOI: 10.1109/TVCG.2010.251 (cit. on p. 27).

Wald, I., S. Woop, C. Benthin, G. S. Johnson, and M. Ernst (2014). „Embree: A Kernel Framework for Efficient CPU Ray Tracing“. In: *ACM Trans. Graph.* 33.4, 143:1–143:8. ISSN: 0730-0301. DOI: 10.1145/2601097.2601199 (cit. on p. 27).

Wang, S. W. and A. E. Kaufman (1993). „Volume sampled voxelization of geometric primitives“. In: *Proceedings of the 4th conference on Visualization*. VIS '93. San Jose, California: IEEE Computer Society, pp. 78–84. ISBN: 0-8186-3940-7. DOI: 10.1109/VISUAL.1993.398854 (cit. on p. 23).

— (1995). „Volume sculpting“. In: *Proceedings of the 1995 Symposium on Interactive 3D Graphics*. I3D '95. ACM, pp. 151–156. ISBN: 0-89791-736-7. DOI: 10.1145/199404.199430 (cit. on p. 37).

Weiler, K. and P. Atherton (1977). „Hidden surface removal using polygon area sorting“. In: *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '77, pp. 214–222. DOI: 10.1145/563858.563896 (cit. on pp. 24, 73).

Weiskopf, D., K. Engel, and T. Ertl (2003). „Interactive clipping techniques for texture-based volume visualization and volume shading“. In: *IEEE Transactions on Visualization and Computer Graphics* 9.3, pp. 298–312. ISSN: 1077-2626. DOI: 10.1109/TVCG.2003.1207438 (cit. on p. 37).

Whittaker, E. T. (1915). „On the Functions which are represented by the Expansions of the Interpolation-Theory“. In: *Proceedings of the Royal Society of Edinburgh* 35, pp. 181–194. ISSN: 0370-1646. DOI: 10.1017/s0370164600017806 (cit. on p. 16).

Williams, D., S. Grimm, E. Coto, A. Roudsari, and H. Hatzakis (2008). „Volumetric Curved Planar Reformation for Virtual Endoscopy“. In: *IEEE Transactions on Visualization and Computer Graphics* 14.1, pp. 109–119. DOI: 10.1109/TVCG.2007.1068 (cit. on p. 35).

Young, P. (2006). *Coverage Sampled Antialiasing*. Tech. rep. NVIDIA Corporation (cit. on pp. 25, 143).

Zhang, L., W. Chen, D. S. Ebert, and Q. Peng (2007). „Conservative voxelization“. In: *The Visual Computer* 23.9, pp. 783–792. ISSN: 1432-2315. DOI: 10.1007/s00371-007-0149-0 (cit. on pp. 23, 37).

Curriculum Vitae

ThomasAuzinger

Doctoral Researcher

contact

Favoritenstrasse 9-11
1040, Wien
Austria

+43 1 58801-18683
+43 1 58801-18698

auzinger@cg.tuwien.ac.at
thomasauzinger.html
li://thomasauzinger

languages

native German
professional English
fluent Spanish

interests

computer graphics
fabrication
parallel computing

nationality

Austria

mission

Thomas Auzinger is research assistant and doctoral student at the Institute of Computer Graphics and Algorithms of the Vienna University of Technology. Under the supervision of Assoc. Prof. Michael Wimmer, he investigates fundamental and applied research questions in rendering, visualization, modeling, and fabrication.

education

2015 *(exp)* — **Doctor** of Technical Sciences — Vienna University of Technology
Sampled and Prefiltered Anti-Aliasing on Parallel Hardware

2015 *(exp)* — **Diploma Supplement** on Innovation — Informatics Innovation Center
Multi-semester course on innovation and entrepreneurship

2010 — **Master** of Science (Mathematical Physics) — University of Vienna
Rotating Bose-Einstein Condensates in Partially Anisotropic Traps

2009 — **Bachelor** of Science — University of Vienna
Bose-Einstein Condensates and Vortices

appointments

2009–2015 — **Vienna University of Technology** — Vienna, Austria
Doctoral Researcher
Focus on rendering, GPGPU, and medical visualization

2001–2009 — **Austrian Institute of Technology – Health Physics Group** — Seibersdorf, Austria
Software Developer and *Technical Assistant*
Data acquisition, hardware control, simulation, data evaluation,…

projects

2012–2016 — **Modern Functional Analysis in Computer Graphics** — Austrian Science Fund (FWF)
Research Assistant

2008–2012 — **Detailed Surfaces for Interactive Rendering** — Austrian Science Fund (FWF)
Research Assistant

teaching

2014 — **Invited Lecturer** — Universidad de las Ciencias Informáticas, Cuba
Course on *General-Purpose Programming on Graphic Units*

2010–Now — **Lecturer** — Vienna University of Technology
Master's course on *Rendering* (with Károly Zsolnai)

2010–Now — **Student Supervision** — Vienna University of Technology
Supervision of Bachelor and Master theses and lab exercises

activities

2014–Now **Volunteer and Sole Responsible** Faculty of Informatics, Vienna University of Technology
Design, development and maintenance of LaTeX template for all faculty theses

2014–Now **Invited Speaker** Universidad de Jaén (ESP), Czech Technical University (CZE)
Talks on personal research

2013 **Presenter (Best Poster Award)** PUMPS Summer School, Barcelona, Spain
Course participant and poster presenter

2010-2011 **Main Organizer** Vienna, Austria
Russian-Austrian Bilateral Scientific Seminar on Visual Computing

2010–Now **Reviewer** Eurographics, CGF, CGI, TVCJ,…
Journals and conferences in the field of Visual Computing

2010–Now **Member** ACM, Eurographics, IEEE
Professional membership

publications

journals

Layer-based Procedural Modeling of Facades
Martin Ilčík, Przemyslaw Musialski, **Thomas Auzinger**, Michael Wimmer
Computer Graphics Forum (2015). 2015

Separable Subsurface Scattering
Jorge Jimenez, Károly Zsolnai, Adrian Jarabo, Christian Freude, **Thomas Auzinger**, Xian-Chun Wu, Javier Pahlen, Michael Wimmer, Diego Gutierrez
Computer Graphics Forum (2015). 2015

Guided Volume Editing based on Histogram Dissimilarity
Alexey Karimov, Gabriel Mistelbauer, **Thomas Auzinger**, Stefan Bruckner
Computer Graphics Forum (2015). 2015

Partial Shape Matching Using Transformation Parameter Similarity
Paul Guerrero, **Thomas Auzinger**, Michael Wimmer, Stefan Jeschke
Computer Graphics Forum (2014). 2014

Analytic Visibility on the GPU
Thomas Auzinger, Michael Wimmer, Stefan Jeschke
Computer Graphics Forum 32.2pt4 (May 2013) pp. 409–418. 2013

Vessel Visualization using Curved Surface Reformation
Thomas Auzinger*, Gabriel Mistelbauer*, Ivan Baclija, Rudiger Schernthaner, Arnold Kochl, Michael Wimmer, M. Eduard Groller, Stefan Bruckner
IEEE Transactions on Visualization and Computer Graphics 19.12 (2013) pp. 2858–2867. 2013

Analytic Anti-Aliasing of Linear Functions on Polytopes
Thomas Auzinger, Michael Guthe, Stefan Jeschke
Computer Graphics Forum 31.2pt1 (May 2012) pp. 335–344. 2012

conferences

YMCA - Your Mesh Comparison Application
Johanna Schmidt, Reinhold Preiner, **Thomas Auzinger**, Michael Wimmer, Meister Eduard Gröller, Stefan Bruckner
IEEE Visual Analytics Science and Technology, 2014, Paris, France

General-Purpose Graphics Processing Units in Service-Oriented Architectures
María Carmen Calatrava Moreno, **Thomas Auzinger**
Proceedings of the 6th IEEE International Conference on Service Oriented Computing and Applications, 2013, Kauai

Non-Sampled Anti-Aliasing
Thomas Auzinger, Przemyslaw Musialski, Reinhold Preiner, Michael Wimmer
Proceedings of the 18th International Workshop on Vision, Modeling and Visualization, 2013, Lugano, Switzerland

GeigerCam: Measuring Radioactivity with Webcams
Thomas Auzinger, Ralf Habel, Andreas Musilek, Dieter Hainz, Michael Wimmer
ACM SIGGRAPH 2012 Posters, 2012, Los Angeles, California