New Models for High-Quality Surface Reconstruction and Rendering

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
von

Dipl. Inform. Thomas Kalbe
geboren in Groß Gerau, Deutschland

Referenten der Arbeit:	Prof. Dr. techn. Dieter W. Fellner Technische Universität Darmstadt Prof. Dr. Ing. Holger Theisel AG Visual Computing, Universität Magdeburg
Tag der Einreichung:	18. Oktober 2010
Tag der mündlichen Prüfung:	8. März 2011

D 17
Darmstadt 2011

Neue Modelle zur Rekonstruktion und Visualisierung von Oberflächen mit hoher Qualität

Kurzzusammenfassung

Die effiziente Rekonstruktion und artefaktfreie Visualisierung von Oberflächen aus gemessenen Daten realer Objekte ist in einer Vielzahl von Anwendungen von Wichtigkeit. Beispielhaft seien medizinische oder wissenschaftliche Visualisierungen, Qualitätskontrolle von Bauteilen, oder Anwendung in Computerspielen, Film und Fernsehen genannt. Der Hauptbeitrag dieser Arbeit besteht in der Entwicklung der ersten effizienten GPU-basierten Rekonstruktions- und Visualisierungsmethoden basierend auf *trivariaten Splines*, das sind Splines definiert bezüglich geeigneter Tetraederzerlegungen des Raumes. Wir zeigen, dass sich diese Splines sehr gut zur Rekonstruktion und Visualisierung von Oberflächen aus Volumendaten in Echtzeit eignen, wobei die Oberflächen mit hoher visueller Qualität, also unter Vermeidung visueller Artefakte, angezeigt werden. Wir entwickeln zudem einen neuen quasi-interpolierenden Operator, der auf einen C^1-differenzierbaren Spline mit Polynomen vom Grad zwei führt. Dies ist die erste direkte Methode, um 3D Daten mit C^1-differenzierbaren Polynomen vom Grad zwei zu approximieren, ohne dass Tetraeder unterteilt werden müssen. Weiterhin entwickeln wir eine neue Projektionsmethode zur Triangulierung von unstrukturierten Punktemengen, die im Vergleich zu bestehenden Methoden hochqualitative Dreiecksnetze mit einer garantierten numerischen Stabilität generiert.

Zusammenfassung

Die Rekonstruktion und Visualisierung von Oberflächen aus gemessenen Daten ist ein aktives Forschungsgebiet mit einer Vielzahl von Anwendungen. Im medizinischen Bereich sind bildgebende Verfahren wie CT oder MRI in der Diagnose nicht mehr wegzudenken. Diese Verfahren liefern skalare Messwerte auf regulären dreidimensionalen Gittern, sogenannte *Volumendaten*. Weitere Einsatzgebiete von Volumendaten finden sich auch in der industriellen Qualitätskontrolle, der strukturellen Biologie, sowie der Archäologie und Geologie, etwa zum Auffinden von Ölfeldern. Einige Systemen zur Modellierung von Freiformflächen im *Computer Aided Geometric Design* (CAGD) erzeugen ebenfalls Volumendaten als Zwischenrepräsentation, aus denen die gesuchten Oberflächen extrahiert werden müssen. Weiterhin fallen synthetische Volumendaten auch als Ergebnis von numerischen Simulationen an. Eine zweite Klasse von Daten sind durch Laser-Abtastung realer Objekte gewonnene Mengen diskreter 3D Punkte. Verfahren, die aus diesen *unstrukturierten Punktemengen* Oberflächen rekonstruieren, finden insbesondere Anwendung in industriellen Fertigungsprozessen, beispielsweise der Automobilindustrie, aber auch in der Bewahrung von Objekten hoher kultureller Bedeutung, etwa Gebäude oder Statuen.

Eine der an moderne Verfahren zur Oberflächenrekonstruktion gestellten Anforderungen ist hohe visuelle Qualität, also eine möglichst artefaktfreie Darstellung der rekonstruierten Objekte. Gleichzeitig sollte die Rekonstruktion in einer vertretbaren Geschwindigkeit erfolgen. Eine der Schwierigkeiten dabei ist es, dass moderne Abtastverfahren sehr große bis riesige Datenmengen erzeugen. Im medizinischen Bereich sind Volumendaten mit Gittergrößen von 512^3 Punkten heute üblich. Die im *Digital Michelangelo Project* abgetasteten Objekte bestehen aus mehr als hundert Millionen Punkten. Rekonstruktionsverfahren stellen somit hohe Anforderungen an die Hardware, sowohl in Bezug auf benötigte Rechenzeit sowie Speicherverbrauch. Desweiteren ist der Scan-Prozess nicht hundertprozentig exakt. Abhängig vom verwendeten Verfahren, den äusseren Bedingungen, und dem zu scannenden Objekt können die erzeugten Datensätze erhebliches Rauschen und Ungenauigkeiten aufweisen. Es ist daher für jede Rekonstruktionsmethode von Bedeutung, dass Rauschen berücksichtigt wird, dabei aber nicht zu viele Detailinformationen verloren gehen.

In dieser Arbeit werden neue Rekonstruktionsverfahren und Visualisierungsmethoden für gemessene Daten vorgestellt und analysiert. Der erste Teil der Arbeit befasst sich mit Rekonstruktionsverfahren für Oberflächen aus Volumendaten, wobei wir einen relativ neuen Typ von *Splines* (stückweise zusammengesetzte Polynome) einsetzen, nämlich trivariate Splines bezüglich geeigneter Tetraederzerlegungen des Raumes. Diese Splines besitzen einige Vorteile gegenüber Standardverfahren, wie etwa Tensorprodukt-Splines, aber auch eine deutlich komplexere Struktur, der für einen effizienten Rendering-Ansatz

Rechnung getragen werden muss. Im zweiten Teil befassen wir uns mit der Oberflächenrekonstruktion aus unstrukturierten Punktemengen, wobei wir einen projektiven Ansatz zur Generierung von Dreiecksnetzen hoher Qualität einsetzen. Ein Maß für die Qualität eines Dreiecksnetzes ist hierbei hohe Regularität (die meisten Eckpunkte besitzen den Kantengrad sechs), die Form der Dreiecke (möglichst gleichschenklige Dreiecke), und Anpassung an lokale Oberflächenstruktur. Unser Hauptbeitrag hierbei ist eine verbesserte Projektionsmethode, die deutlich stabiler und effizienter als vorherige Methoden arbeitet.

Rekonstruktion und Visualisierung von Oberflächen aus Volumendaten

Für die Visualisierung von Volumendaten ist es von zentraler Bedeutung, geeignete mathematische Modelle zu entwickeln, die die Daten an den Gitterpunkten interpolieren oder approximieren, und kontinuierlich zwischen den Gitterpositionen variieren. Ausgehend von den praktischen Erfordernissen effizienter Volumenvisualisierung, wie schnelle Berechnung und Auswertung der Modelle, beweisbare Approximationseigenschaften für die Werte sowie Ableitungen, und Unempfindlichkeit gegenüber Rauschen, werden heute vor allem polynomiale Filter für die Interpolation und Approximation diskreter Volumendaten eingesetzt. Die meisten bekannten Ansätze basieren hierbei auf Tensorprodukt-Splines in drei Variablen, wobei die prominentesten Vertreter die trilinear stetigen, sowie triquadratisch und trikubisch differenzierbaren, Splines sind. Aufgrund seiner Einfachheit kommt das trilineare Modell heute sehr häufig in Systemen zur Echtzeit-Volumenvisualisierung zum Einsatz. Die Visualisierungen sind jedoch nicht glatt und sichtbare und störende Artefakte, beispielsweise bei der Beleuchtung oder an den Silhouetten, sind fast unvermeidbar. Triquadratische und trikubische Splines reduzieren diese Artefakte und haben ein glatteres Erscheinungsbild, führen aber auf Polynome in drei Variablen vom totalen Grad sechs beziehungsweise neun. Neben dem damit verbundenen deutlich höheren Aufwand für die Auswertung der Modelle sowie der Ableitungen ist eine exakte Schnittberechnung mit den Sichtstrahlen bei Strahlverfolgungsverfahren (Ray Casting) nicht möglich. Stattdessen müssen beispielsweise Intervall-Verfeinerungsmethoden angewandt werden, die insbesondere an den Objektsilhouetten aufwendig sind.

Vor kurzem wurde erstmalig vorgeschlagen, eine neue Klasse von Splines als Modelle zur Visualisierung zu verwenden, dies sind stückweise differenzierbar zusammengesetzte Polynome in drei Variablen definiert bezüglich geeigneter Tetraederzerlegungen des Raumes. Die Vorteile dieser trivariaten Splines bezüglich der Visualisierung von Iso-Oberflächen aus Volumendaten mittels Ray Casting sind

- minimaler totaler Grad der polynomialen Stücke
- exakte und effiziente Berechnung der Oberflächenschnitte durch Lösung quadratischer und kubischer Gleichungen
- optimale Approximation der Ableitungen

- direkte Anwendbarkeit von Standardwerkzeugen der CAGD und Bernstein-Bézier-Techniken zur stabilen und effizienten Auswertung
- schnelles Überprüfen, ob ein gegebenes Polynomstück zur sichtbaren Oberfläche beiträgt
- vernachlässigbar geringer Zusatzaufwand zur Bestimmung der Normalen

Genauer verwenden wir sogenannte *Quasi-Interpolationsmethoden*, die sich dadurch auszeichnen, dass die Koeffizienten der Polynomstücke direkt durch geeignete Mittelungen der Volumendaten in einer lokalen Nachbarschaft verfügbar sind. Bis heute sind nur wenige Splines dieser Bauart bekannt, denn das Finden geeigneter Tetraederpartitionen und Mittelungsoperatoren für die Koeffizienten, so dass eine ausreichende Approximationsgüte erreicht wird, ist insbesondere für niedrige Polynomgrade nicht trivial. Wir verwenden *quadratische Super-Splines* und kubische C^1 Splines bezüglich *Typ-6* Tetraderzerlegungen, wobei jeder Datenwürfel des Volumengitters in vierundzwanzig kongruente Tetraeder zerlegt wird. Weiterhin entwickeln wir einen neuen Quasi-Interpolanten, der erstmalig das Problem einer lokalen Approximationsmethode von 3D Daten mittels Polynomen vom Grad zwei bei gleichzeitiger Differenzierbarkeit an den Übergängen löst. Dieser Spline basiert auf einer Zerlegung des Raumes in abgeflachte Oktaeder (truncated octahedra, TO), die weiter in zwei Klassen von Tetraedern zerlegt werden. Wir zeigen, dass dieser neue Spline die Werte an den Gitterpunkten mit beinahe optimaler Ordnung approximiert, und eine optimale Approximationsordnung für die Gradienten erreicht. Ein weiterer Vorteil der verwendeten Operatoren ist es, dass die zur Mittelung benötigten lokalen Nachbarschaften sehr klein sind. Beispielsweise basiert der kubische C^1 Spline auf einer Nachbarschaft der 23 nächstgelegenden Werte, der quadratische C^1 Splines benötigt 28 benachbarte Werte. Zum Vergleich sei erwähnt, dass trikubische Tensorprodukt-Splines auf einer Mittelung von 64 Werten basieren.

Quasi-Interpolationsmethoden wurden bereits zur Visualisierung mittels Ray Casting erfolgreich eingesetzt, wobei gezeigt wurde, dass die Methoden das Potential besitzen, realistische, natürlich aussehende und nahezu artefaktfreie Bilder zu erzeugen. Dies ist insbesondere in der Medizin, aber auch in der Qualitätskontrolle von Bauteilen von zentraler Bedeutung. Die ersten Ansätze zur Visualisierung dieser Splines waren zunächst noch Software-basiert, wobei nur niedrige Bildraten erreicht werden konnten. In den letzten Jahren wurden dagegen programmierbare Grafikprozessoren (GPUs) immer leistungsfähiger und flexibler, wodurch eine Vielzahl von Ansätzen zur Echtzeit-Visualisierung von Volumendaten mittels GPUs entwickelt wurden. Auch für Splines auf Tetraederzerlegungen wurden Methoden zur Visualisierung mittels GPUs vorgeschlagen, die jedoch verschiedene ungelöste Schwierigkeiten aufwiesen. Beispielsweise war der Speicherverbrauch für die Polynomkoeffizienten und die Hüllgeometrien (Tetraeder) sehr hoch, wodurch im Vergleich zu Standardmethoden nur geringe Bildraten erreicht wurden und nur kleine Datensätze überhaupt visualisiert werden konnten.

Ein Hauptbeitrag dieser Arbeit ist es, hoch effiziente Ansätze zur Visualisierung von Oberflächen mittels GPUs basierend auf trivariaten Splines zu entwickeln. Wir gehen dabei zunächst auf signifikante Verbesserungen der bekannten Ansätze ein, die auf einer

Projektion der Tetraeder auf den Bildschirm basieren. Wir zeigen hierbei, dass es von großer Wichtigkeit ist, die Struktur der Splines für eine effiziente Visualisierung zu berücksichtigen, um den Speicherverbrauch und die arithmetische Komplexität der Berechnungen zu reduzieren. Unsere Ergebnisse zeigen eine deutlich verbesserte Performanz gegenüber vorherigen Methoden. Gleichzeitig zeigen wir aber auch die Grenzen dieser Projektionsmethoden auf derzeitigen GPUs auf, da für praxisrelevante Datensätze viele Millionen Tetraeder entstehen, und jeder dieser zur Oberfläche beitragende Tetraeder separat behandelt werden muss.

Wir verfolgen daher einen weiteren, alternativen Ansatz zur Volumenvisualisierung mittels trivariater Splines. Hierbei werden nicht mehr einzelne Tetraeder projiziert, sondern es findet eine Strahltraversierung durch das gesamte Volumen statt. Wir entwickeln hierfür effiziente Algorithmen zur Auslassung leerer Bereiche (*empty space skipping*) sowie zur Traversierung der zugrundeliegenden gleichmäßigen Tetraederpartitionen, wobei wir auch hierbei von den Vorteilen trivariater Quasi-Interpolanten profitieren (schnelle Berechnung der Polynomstücke, kleine Nachbarschaften, effiziente Schnittberechnungen, etc.). Wir zeigen, dass in diesem bildbasierten Ansatz die Performanz beinahe unabhängig von der Anzahl der auf der Oberfläche liegenden Tetraeder ist, und wir somit auch große, praxisrelevante Datensätze mit hohen Bildraten anzeigen können. Weiterhin zeigen wir, dass neue Programmieransätze auf GPUs jenseits der bekannten Grafik-Pipeline, wie NVidia's CUDA, zu unserem Vorteil ausgenutzt werden können. Im Vergleich zur gewöhnlichen Grafik-Pipeline kann durch den Einsatz von CUDA auf derselben Hardware-Plattform eine Geschwindigkeitssteigerung um den Faktor drei erreicht werden. Unsere Ergebnisse zeigen somit, dass trivariate Quasi-Interpolationsmethoden für Echtzeit-Visualisierungen großer Datensätze mit hoher visueller Qualität auf heutiger Grafik-Hardware sehr gut geeignet sind.

Dreiecksnetzrekonstruktion aus unstrukturierten Punktemengen

Neben der Rekonstruktion und Visualisierung von Oberflächen aus Volumendaten befassen wir uns in dieser Arbeit weiterhin mit der Erzeugung von Dreiecksnetzen aus unstrukturierten Punktemengen. Diese Punktemengen werden üblicherweise durch 3D Laser-Abtastung realer Objekte gewonnen, wobei jedoch in der Praxis gewisse Ungenauigkeiten, wie Messrauschen oder Selbstverdeckung, auftreten können. In der Literatur sind bereits eine Vielzahl von Verfahren zur Erzeugung von Oberflächen aus gemessenen Punkten bekannt, die verschiedene Vor- und Nachteile besitzen, wie beispielsweise Empfindlichkeit gegenüber Messrauschen, korrekte Behandlung von Rändern und Löchern, Kontrolle über Form und Anzahl der erzeugten Oberflächenprimitive, Anpassung an lokale Oberflächenstruktur, Behandlung von scharfen Kanten, oder Geschwindigkeit und Stabilität der Netzerzeugung.

Wir konzentrieren uns hier auf *Projektionsmethoden*, die ausgehend von einem bereits triangulierten Bereiches, der *Front*, weitere Punkte durch eine Heuristik voraussagen und auf eine lokale Oberflächenapproximation projizieren. Der wesentliche Vorteil

dieser als *Advancing Front* bekannten Methoden ist die hohe Qualität der erzeugten Triangulierungen. Die entstehenden Dreiecke sind zumeist annähernd gleichseitig und die Netze weisen eine hohe Regularität auf, d.h. die Mehrzahl der Eckpunkte besitzen die Kantenvalenz sechs (ca. 75% der Eckpunkte in unseren Tests), die Valenzen der übrigen Eckpunkte sind meist fünf oder sieben. Algorithmen zur späteren Nachbearbeitung wie Subdivision-Verfahren oder Netzkompression, profitieren im hohen Maße von einer Regularität der Dreiecksnetze. Die Form der Dreiecke hat direkten Einfluss auf die Stabilität bei numerischen Approximationsverfahren. Weiterhin lässt sich der Detaillierungsgrad der Dreiecksnetze steuern, und die Größe der Dreiecke passt sich dynamisch an die lokale Oberflächenstruktur an, so dass in Bereichen mit hoher Krümmung mehr Dreiecke erzeugt werden, während annähernd flache Bereiche mit vergleichsweise wenigen Dreiecken approximiert werden.

Von zentraler Bedeutung für Advancing Front Algorithmen ist die Bereitstellung eines effizienten und robusten Projektionsverfahrens zur Abbildung der vorhergesagten Punkte auf die lokale Approximation der Oberfläche, wobei in den letzten Jahren das *Moving Least Squares* (MLS) Projektionsverfahren große Popularität erreicht hat. Wie unsere Untersuchungen und gewisse Hinweise in der Literatur gezeigt haben, ist ein schwerwiegendes Problem von MLS jedoch, dass die Robustheit des Verfahrens nicht immer garantiert werden kann. Die Ursache liegt in der Notwendigkeit zur numerischen Lösung eines nicht-linearen Minimierungsproblems, bei dem die üblichen Schwierigkeiten bei der Suche nach einem lokalen Minimum, das bestimmte Randbediungen erfüllen muss, auch tatsächlich in der Praxis auftreten. Wir entwickeln in dieser Arbeit daher ein alternatives Projektionsverfahren, das sich gegenüber MLS durch deutlich bessere Rekonstruktionseigenschaften und schnellere Berechnungen der Projektionen auszeichnet. Im Gegensatz zu Projektionen mittels MLS kommt unser neues Projektionsverfahren ohne die numerische Lösung nicht-linearer Optimierungs- und verwandter Probleme aus. Wir zeigen, dass zur Durchführung des neuen Verfahrens lediglich Berechnungen benötigt werden, für welche robuste und effiziente Algorithmen zur Verfügung stehen.

Unser Projektionsverfahren basiert auf einer 3D Delaunay-Triangulierung zur Ermittlung der Konnektivität der gegebenen Punkte in einem Vorverarbeitungsschritt. Dies hat zum Einen den Vorteil, dass wir die Punkte in einer lokalen Nachbarschaft nach der approximierten Riemann'schen Distanz gewichten können, anstelle wie bei MLS auf die Euklidische Distanz angewiesen zu sein, was häufig zu Fehlprojektionen führt. Der zweite Vorteil ist, dass geschätzte Normalen anfallen, die wir zu einer ersten Ausrichtung der für die Projektion benötigten lokalen Koordinatensysteme verwenden können. Im Gegensatz zu MLS funktioniert unser Verfahren daher auch dann besonders gut, wenn keine gemessenen Normalen vorliegen. Die Oberflächen werden lokal durch bivariate Polynome approximiert, wobei wir im Gegensatz zu MLS einen variablen Polynomgrad verwenden. Die Idee dahinter ist es, dass Polynome von höherem Grad die Oberfläche und die Krümmung besser approximieren, es aber zu Situationen kommen kann, in denen die Polynomapproximation instabil wird (eine kleine Veränderung der Punkte führt zu einem komplett anderen Polynom). In diesen Fällen reduzieren wir den Polynomgrad und beginnen die Approximation erneut.

Unsere Ergebnisse zeigen, dass das neue Projektionsverfahren deutlich robuster ist als

MLS und auch komplexe Oberflächenstrukturen zu rekonstruieren in der Lage ist. Die Anpassung des Polynomgrads führt zudem auf verbesserte Triangulierungen, insbesondere in Regionen mit hoher Krümmung.

Erklärung zur Dissertation

Hiermit versichere ich die vorliegende Dissertation selbständig nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 31.1.2011 Thomas Kalbe

Abstract

The efficient reconstruction and artifact-free visualization of surfaces from measured real-world data is an important issue in various applications, such as medical and scientific visualization, quality control, and the media-related industry. The main contribution of this thesis is the development of the first efficient GPU-based reconstruction and visualization methods using trivariate splines, i.e., splines defined on tetrahedral partitions. Our methods show that these models are very well-suited for real-time reconstruction and high-quality visualizations of surfaces from volume data. We create a new quasi-interpolating operator which for the first time solves the problem of finding a globally C^1-smooth quadratic spline approximating data and where no tetrahedra need to be further subdivided. In addition, we devise a new projection method for point sets arising from a sufficiently dense sampling of objects. Compared with existing approaches, high-quality surface triangulations can be generated with guaranteed numerical stability.

Keywords. Piecewise polynomials; trivariate splines; quasi-interpolation; volume data; GPU ray casting; surface reconstruction; point set surfaces

Acknowledgements

I would like to thank the following people for supporting this thesis (in alphabetical order): Prof. D. Fellner, Simon Fuhrmann, Prof. Michael Goesele, Thomas Koch, Arjan Kuijper, Alexander Marinc, Markus Rhein, Christian Rössl, Tania Sorokina, Stefan Uhrig, Dominik Wodniok, and Frank Zeilfelder.

For my beloved wife Jessica.

In memory of my parents.

Contents

Chapter 1

Introduction

The reconstruction and visualization of surfaces from a variety of data sources has many applications and is thus an active field of research. We can classify measured data according to its sampling as either *uniform* (structured) or *non-uniform* (unstructured, scattered). Scanning devices such as Computer Tomography (CT), Magnetic Resonance Imaging (MRI), Positron Emission Tomography (PET) or 3D ultrasound produce scalar data on regular volumetric (i.e., three-dimensional) grids and are important tools in medical research and diagnosis, and in many fields of engineering. In structural biology, data obtained from Cryo-Electron Tomography (CET) can be used to deduce the 3D shape of cell structures and viruses. The type of data associated with the grid points typically reflects the density of the scanned objects or the distinct reaction of the atomic configuration of tissue when exposed to, e.g., magnetic fields or nuclear radiation. Synthetic data may be generated for example from numerical simulations, or signed-distance functions taken from polyhedral meshes. We refer to data sampled on a regular volumetric grid as *volume data*. The visualization of this data is a common task in various applications. Examples are medical imaging, scientific visualization, industrial quality control, geology, archaeology, reverse engineering, Computer Aided Geometric Design and the visual media (TV, cinema, and computer games). Here, we are interested in *efficient methods* leading to *high-quality visualizations* of the reconstructed objects, for instance, human organs or machinery parts.

A different source of data is typically generated from laser range scanners as a dense sampling of real-world objects. These scanners generate a set of 3D positions of discrete points (*unstructured point sets* or *point clouds*) scanned from the object. Sometimes, we also have an estimate of the normal, color, or reflection characteristics for each point. Applications for this kind of data arise in, e.g., the automotive industry, architecture, or archaeology. It is possible to generate a volumetric representation of point set data for example by computing *signed distance functions* (SDF) [HDD*92, SOM04, JBS06] and then apply suitable reconstruction and visualization methods to the arising data on a regular 3D grid, but usually we want methods that directly construct surfaces from unstructured points.

Generally, the obtained data sets can be huge: in the process of oil-finding overlapping volume data sets of several GBytes size are generated; in the *Digital Michelangelo Project* [LPC*00], the high-resolution scan of the David statue consists of over hundred of million points. Surface reconstruction and visualization methods from real-world data are thus computationally intensive and demand a substantial amount of memory for the

processing of the data. In addition, the scanning process is not necessarily accurate. Depending on the particular method and conditions of the sampling substantial amounts of *noise* may be generated. It is thus important for surface reconstruction methods to be insensitive to noise while at the same time no relevant information (i.e., surface detail) should get lost.

A primary problem in this area is to develop appropriate continuous models of the measured data which fit our needs in terms of visual quality as well as computational costs for the display and preprocessing of the data. Ideally, the approach involves *real-time reconstruction*: using a standard PC, the non-discrete model is immediately available from the (typically huge) data sets while a certain approximation order is guaranteed, i.e., errors decrease with a certain factor for increasing numbers of data points. Another goal is that the continuous models provide high-quality. Independent from the viewer's position visualizations should appear natural and overall smooth while disruptive artifacts (imperfect silhouettes, undesired oscillations, stair-casing, etc.) are avoided. Further, we demand that standard techniques from *Computer Graphics* (texturing, shading, lighting, etc.) can be applied without too much effort. In this connection, simultaneous approximation of partial derivatives is important since it allows direct sampling of the gradients for complex texturing algorithms or illumination computations. A crucial issue for the visualization of the reconstructed objects is always *interactivity* with the user since, in practice, the reconstructed objects should appear on high-resolution displays without delays.

Volume Visualization

A standard approach for the visualization of regular volumetric data is to use trilinear interpolation [Baj99, ML94] where the tensor-product extension of univariate linear splines interpolating at the grid points results in piecewise cubic polynomials. A sufficiently smooth function is approximated with order two, but in general, reconstructions are not smooth and visual artifacts arise. However, the simplicity of this model has motivated its widespread use. Alternatively, higher-order interpolation such as triquadratic, or tricubic tensor-product splines can be used to construct smooth models. These splines lead to piecewise polynomials of higher total degree, namely six and nine, which are thus more expensive to evaluate.

In this work, we use smooth *trivariate splines*, i.e., piecewise polynomials of low and lowest possible degree (namely, 2 and 3) on uniform tetrahedral partitions connected via smoothness conditions. Since we are interested in real-time reconstructions our models are based on quadratic and cubic *quasi-interpolating splines* in *Bernstein-Bézier form* (B-form) [NRSZ05, SZ05, SZ07b], where the coefficients are simply set as local averages of the uniform data. This is done with care such that approximation order two is guaranteed for the model as well as the partial derivatives while smoothness conditions between neighboring tetrahedral elements are satisfied simultaneously. Further, we develop a new quasi-interpolating operator, which for the first time solves the problem of finding a globally C^1-smooth quadratic spline approximating local data where no tetrahedra need to be further subdivided [RK09].

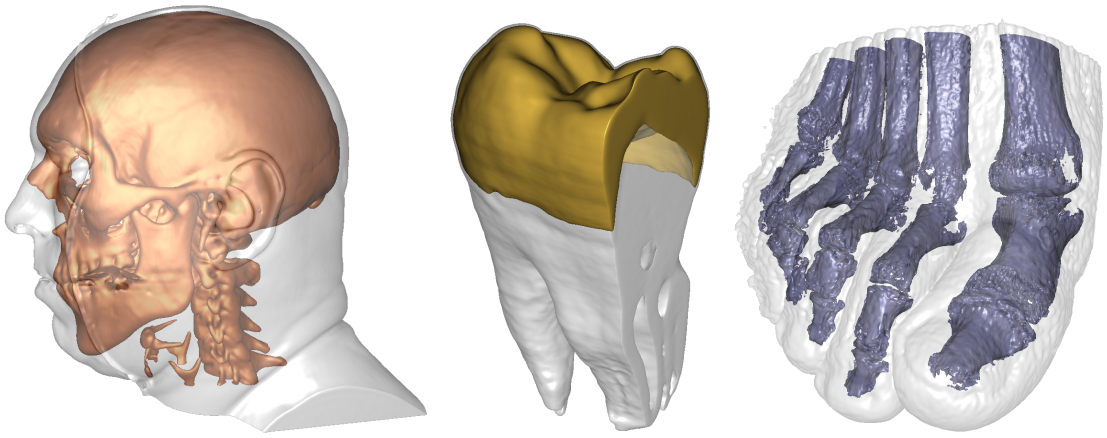

Figure 1.1: Blending of different isosurfaces reconstructed from real-world data sets using smooth trivariate cubic splines . *From left to right*: *VisMale* (256^3 voxels), *Tooth* ($256^2 \times 161$ voxels), and *Foot* (256^3 voxels). VisMale and Tooth are smoothed with a Gaussian filter on the GPU. Choosing the desired isosurfaces and smoothing to an appropriate degree is an interactive process.

Today, highly powerful programmable graphics processing units (GPUs) have become a standard equipment in consumer PCs. Thus, a lot of recent research deals with the development of algorithms for interactive and real-time reconstruction of surfaces and general visualization of data. The main goal of this thesis is to develop the first GPU-based reconstruction and rendering methods for trivariate splines where we use ray casting for interactive high-quality visualizations of varying isosurfaces from volumetric data [KTSZ08, KZ08, KKG09]. For examples of isosurfaces reconstructed using smooth trivariate splines, see Figure 1.1. Here, the low total degree of the polynomials allows for efficient and stable ray-patch intersection tests. Well-known techniques from CAGD, like de Casteljau's algorithm, or *blossoming* can be employed on the GPU for efficient and stable evaluation of the splines. The derivatives needed for direct illumination are immediately available as a by-product of the evaluation. On the other hand, our splines have a more complex structure than those arising from, i.e., trilinear interpolation. Thus, special care has to be taken in order to develop efficient algorithms for reconstruction and visualization with trivariate splines which make optimal use of the limited memory and the high parallelism provided by current GPUs.

Surface Reconstruction from Unstructured Points

Besides surface reconstruction from volume data we also discuss high-quality reconstructions of surfaces from scattered data. 3D laser scanners generate a (sufficiently dense) approximation of object contours as a set of unstructured points. Various approaches that reconstruct a continuous surface from its discrete representation as point sets are known. A common and versatile continuous surface representation are triangle meshes. They are efficient in memory consumption and can be processed with high performance on current graphics hardware. Therefore, the construction of meshes has received a lot of attention

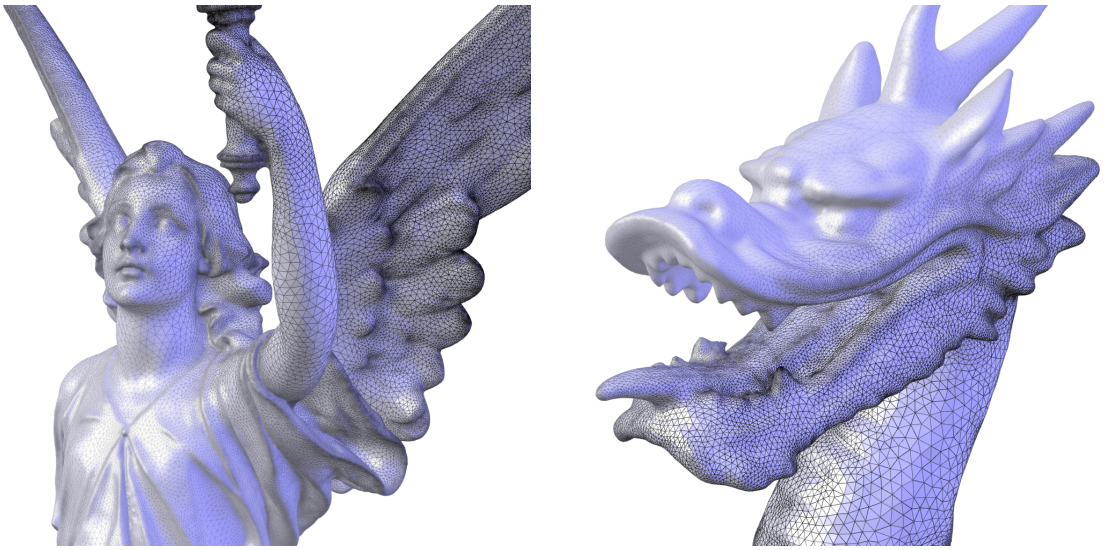

Figure 1.2: Examples of high-quality surface triangulations obtained from our new projection method. *Left*: *Lucy* model, approx. 880 000 triangles. *Right*: *Dragon* head, approx. 390 000 triangles.

with a substantial amount of literature, algorithms and techniques being published. The notion of *quality* of the resulting mesh can refer to the sampling rate, regularity of the mesh, or size and shape of the triangular elements, and is of significance for various applications such as numerical simulations, refinement and subdivision, or mesh compression. A certain type of method for mesh generation depends on a projection which maps newly created vertices onto the surface. These projective methods are especially well suited for a class of algorithms which allow for generation of high-quality meshes that adapt to surface curvature. Mesh granularity can typically be controlled with few user-specified, intuitive parameters. A very popular projective method is the Moving Least Squares (MLS) approach for surface reconstruction [Lev98, ABCO*01, ABCO*03]. The power of MLS surfaces is the ability to naturally cope with input noise in the data.

We show that the MLS approach has some deficiencies and can lead to problems in the computation of the tangent planes and the polynomial approximations of the surface. Additionally, points that are beyond the vicinity of the surface are not guaranteed to be correctly projected. We propose a new projection method based on local surface approximation with polynomials of varying degree [KFU*09]. Unlike MLS, our method does not involve a non-linear optimization or similar problem. Furthermore, we can project points which have an arbitrary distance to the point set. We show that our method is more robust and is able to produce better reconstructions, especially in problematic regions, e.g., surface areas with high curvature. Examples of surface triangulations reconstructed with our projection method are shown in Figure 1.2.

Chapter Overview

The remainder of this thesis is structured as follows:

- In Chapter 2, we give an overview on the current state of the art related to *reconstruction of surfaces* from measured volume data as well as hardware accelerated *interactive visualization* of these data. We conclude this chapter with a review of recent work dealing with smooth trivariate splines for high-quality visualizations and their applications in Computer Graphics.

- In Chapter 3, we lay the theoretical foundations needed for an understanding of the mathematical concepts used in this thesis. This includes the Bernstein-Bézier form (B-form) of the piecewise polynomials which is an ideal description for curves and surfaces. We also describe important tools needed to analyze and evaluate these polynomials, such as the de Casteljau algorithm and blossoming.

- In Chapter 4, we review the literature on (smooth) bivariate and trivariate splines. We further introduce the notation used to describe these spline spaces and conclude with two concrete examples of trivariate splines which are later used in this thesis for efficient and high-quality visualization of sampled data.

- A new spline, which is based on a particular uniform tetrahedral partition of the 3D domain is presented in Chapter 5. For the first time this solves the problem of finding an overall C^1-continuous *quadratic* trivariate spline which approximates local data and where no tetrahedra are further split. We conclude this chapter with an analysis of the approximation properties of the spline.

- In Chapter 6, we give the details of our new GPU implementations for interactive reconstruction and visualization of surfaces from real-world volume data. We show that high-quality renderings of isosurface based on trivariate splines can be done efficiently on modern GPUs. We further evaluate our methods and compare them to related methods, both in terms of visual quality and performance.

- In Chapter 7, we address the problem of finding a surface triangulation which approximates unstructured 3D point sets. We demand our triangulations to fulfill certain *quality criteria* such as shape control over the triangular elements as well as regularity of the triangulation. We further show that our method has improved stability and delivers better triangulations in problematic regions than comparable approaches.

- Finally, we summarize our main results and discuss directions for further research in Chapter 8.

Chapter 2

Reconstruction and Visualization of Volume Data

In this chapter, we give a brief overview of *interactive volume visualization* or *volume rendering*, which is devoted to efficient computational techniques for the visualization of volume data [Baj99, CKY00, Nie00, BW01]. We first discuss filtering functions to obtain a continuous representation of a discretely sampled signal. Next, we consider the most common interactive visualization techniques for scalar data on regular volumetric grids. Volume rendering can be roughly categorized into two general classes. Basically, one distinguishes between *full volume rendering* where the integral equation of a physical emission-transport model has to be numerically solved along rays [Lev90], and *isosurfacing* [ML94, PSL*98, LHJ99, WS01] where surfaces of constant scalar values (called *isosurfaces*) are extracted from the volume model. See Figure 2.1 for examples of both techniques. We can classify isosurfacing further into methods that obtain discrete representations of the surfaces, e.g., triangle meshes. A standard approach in this area is *marching cubes* [LC87] and its variants [KBSS01, Nie04, SW04, NY06]. Alternatively, one is only interested in visualizations of the isosurfaces which is often done by *ray casting* [Lev88]. In this case, the first intersection of each viewing ray with the surface is determined for later illumination. A more detailed discussion on current visualization techniques for interactive volume visualization can be found in the survey by Kaufmann and Mueller [KM05], or the book by Engel et al. [EHKRS06].

We conclude this chapter with an outlook on the particular reconstruction and visualization methods which are presented in this thesis, namely trivariate splines on tetrahedral partitions, and review related previous work.

2.1 Reconstruction Filters for Volume Visualization

In all methods for volume visualization we are interested in finding a continuous function which approximates or interpolates the discrete values given at the grid points. Generally, finding a continuous function from discrete samples can be considered as convolving with a filter function which essentially corresponds to taking weighted averages of the samples. An evaluation of popular reconstruction filters can be found in the surveys by Marschner and Lobb [ML94] and Theußl et al. [TMHG01]. According to the authors, the "best" filter in terms of sampling theory is the *three-dimensional sinc function*. Any band-limited signal can be exactly reconstructed from its discrete samples using the sinc

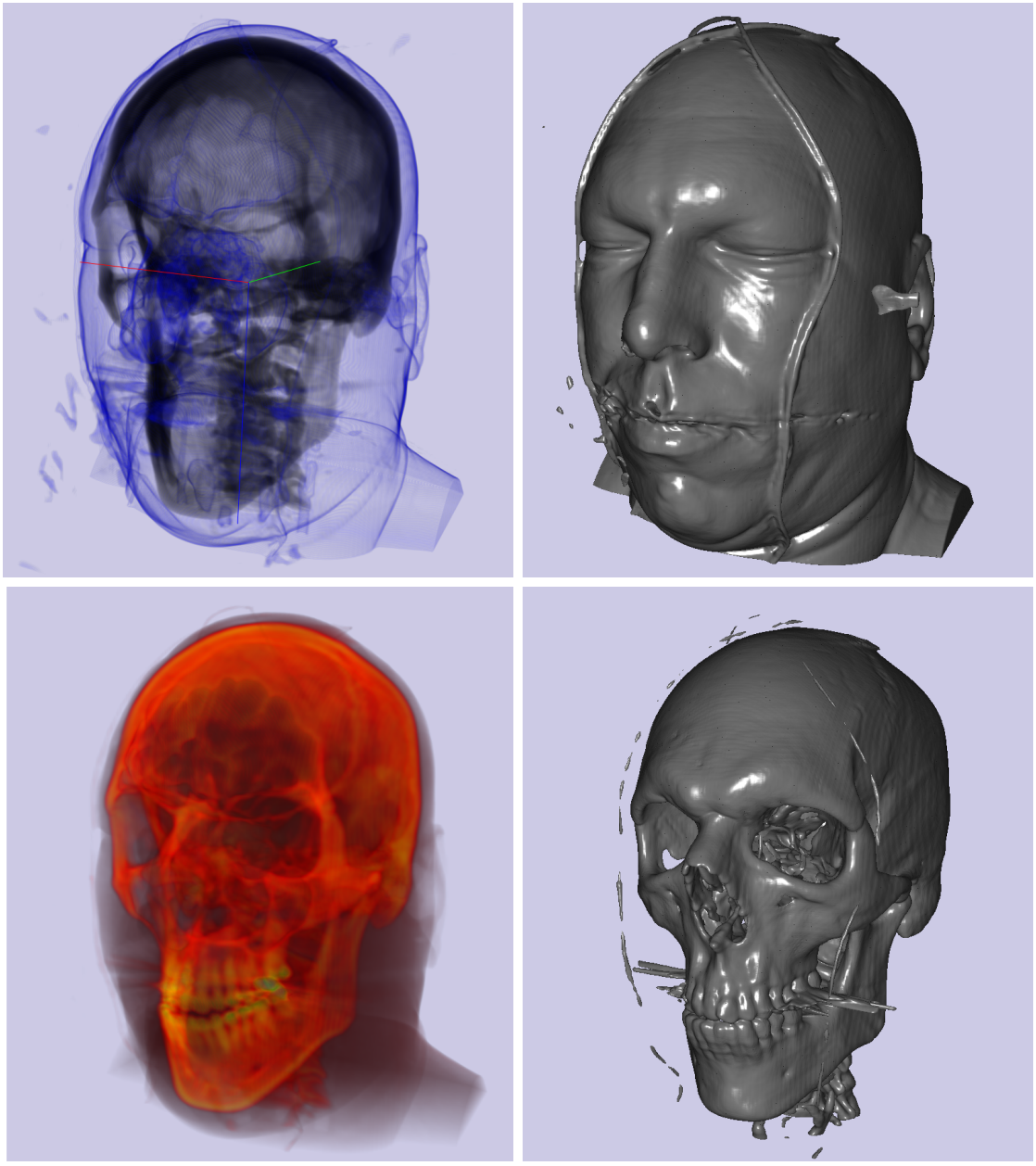

Figure 2.1: Visualizations of a data set obtained from a CT scan. *Left*: full volume rendering. *Right*: isosurfaces

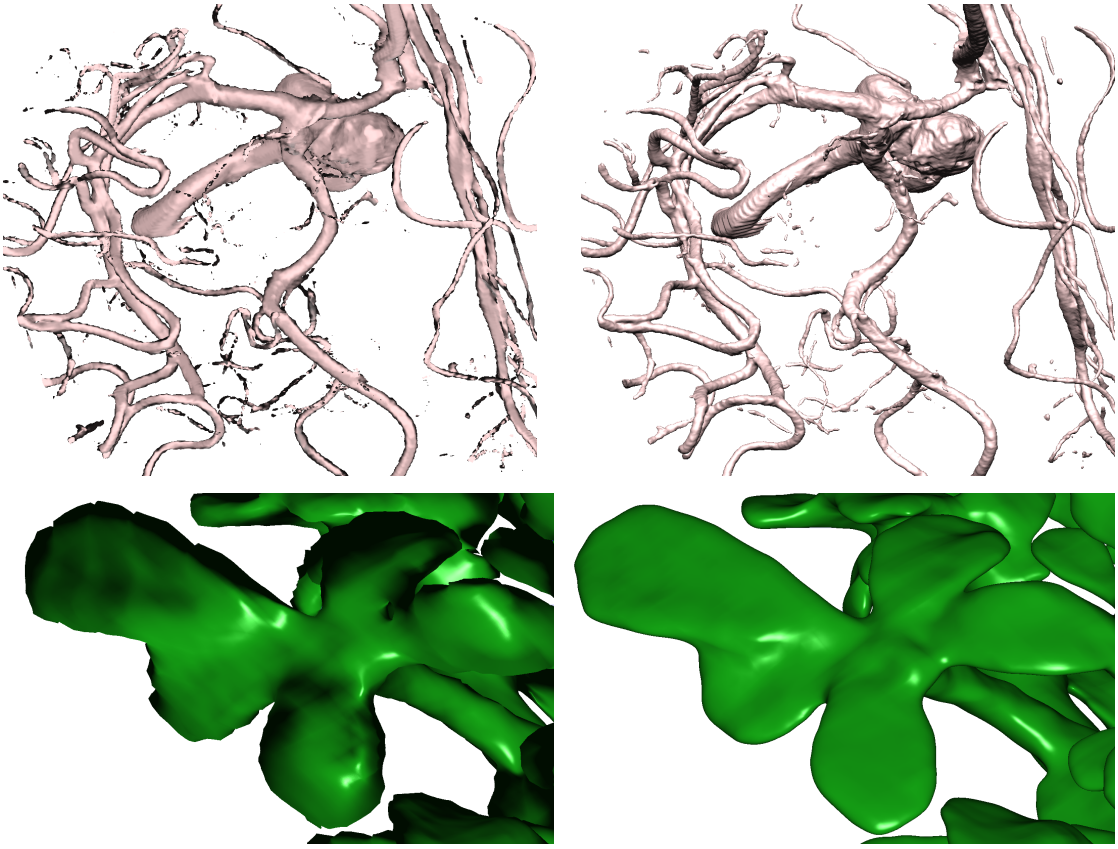

Figure 2.2: Visual comparison of cubic splines. *Left*: trilinear interpolation. *Right*: approximation by smooth cubics.

filter. But, in practice, the sinc filter is difficult to use since it has infinite extend in the space domain.

At the other extreme between reconstruction quality and simplicity of the filter is *trilinear interpolation*, which is known from the marching cubes algorithm [LC87], and leads to continuous piecewise cubic polynomials in three variables obtained from successive univariate linear interpolation along the grid lines. Trilinear interpolation is easy to implement and can be very fast because only the $2 \times 2 \times 2$ nearest samples are considered in the weightings. On the other hand, the visual quality which can be achieved by trilinear interpolation is limited. The gradients are usually obtained from central differences and are thus not smooth and visual artifacts such as aliasing, stair-casing or fringed silhouettes arise. Nevertheless, trilinear interpolation is widespread in practical applications. On the other hand, high quality visualization of volume data requires certain smoothness conditions to be involved (see Figure 2.2). Hence, instead of computing separate derivative models for trilinear interpolation smooth higher-order filter kernels have been proposed to improve the visual quality.

Higher order polynomials are a good compromise between computational complexity

and approximation of the optimal filter. Among these, tricubic *BC-splines* (see [ML94]), which are a generalization of tensor-product B-splines and Catmull-Rom splines, have been proven useful (see for example [HS05]). Thévenaz and Unser [TU01] and Barthe et al. [BMDS02] use approximating C^1-continuous triquadratic B-splines for isosurface reconstruction. The piecewise polynomials of total degree six and nine of the triquadratic and tricubic filters, respectively, increase computational complexity for the evaluation of the polynomials compared with simple trilinear interpolation. This also affects root-finding for ray-surface intersections and derivative computations. Entezari and Möller [EM06] propose trivariate box splines which have the same smoothness and similar reconstruction properties as tricubic B-splines but the kernels have very large support (the $5 \times 5 \times 5$ neighborhood is used) and are thus more smoothing. In addition, evaluation of trivariate box splines is expensive and not always numerical stable [dB93].

Alternatively, truncated Gaussian and windowed sinc filters can be used. Besides the problem of finding suitable window functions which reduce ringing artifacts, these kernels can deliver high quality reconstructions at the expense of an extended support of the filters. As pointed out by, e.g., [ML94], these filters are usually an order of magnitude more expensive to compute than polynomial filters.

A different approach which is investigated in this thesis is based on *trivariate splines* rather than tensor products, see also Chapter 4. These polynomial spaces have been studied in the approximation theory literature of the recent years [Chu89, LS07]. Trivariate splines are piecewise polynomials of low total degree defined on tetrahedral partitions. More specifically, our visualization methods are based on *quasi-interpolation* for quadratic super splines [NRSZ05] and cubic C^1-splines [SZ07b] on *type-6* tetrahedral partitions, as well as quadratic C^1-splines on truncated octahedral partitions [RK09]. Note that these models do not exceed the total polynomial degree of trilinear splines while smoothness conditions are involved. The low degree allows to evaluate the splines efficiently and allows to calculate ray-patch intersections analytically. In addition, the spline is directly available from appropriate weightings in a small and local neighborhood of the centering data value (usually the $3 \times 3 \times 3$ neighborhood is used). Since theses splines rely on the *Bernstein-Bézier form (B-form)* [dB87, Far86, PBP02, Far02] of the piecewise polynomials well-known and approved Bernstein-Bézier techniques from CAGD, such as the de Casteljau algorithm and blossoming [Ram87, Sei93], can be used for stable and efficient evaluation and calculation of ray-surface intersections. Further, the gradients needed for, e.g., illumination of surface points, are almost immediately available as a by-product of the evaluation.

2.2 Hardware-accelerated Volume Rendering

The recent development of programmable graphics processing units (GPUs) has been a massive impulse for interactive volume graphics on consumer hardware [EHKRS06]. The main reason for this development lies in the highly parallel nature of GPUs applicable for large scale local problems. Several techniques for interactive GPU volume rendering have been developed in the recent years. *Cell projection* or *splatting* [Wes90, MJC00, CXZ05]

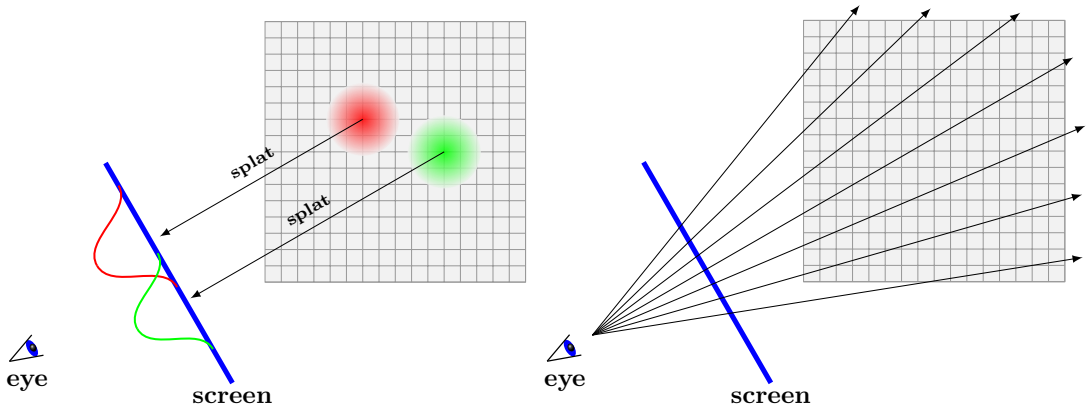

Figure 2.3: *Left*: the splatting principle for volume rendering. For each voxel, the contribution of the kernel is projected onto the viewing plane. The projections are then blended to obtain the final image. *Right*: ray casting. Each viewing ray is traced through the volume grid.

is an object-based visualization method where for each voxel a 3D reconstruction kernel (for example a Gaussian) is projected onto the viewing plane and the contribution of each kernel is blended to obtain a smooth image. An illustration of splatting is given in Figure 2.3, left. An advantage of cell projection is the possibility to apply this technique also to non-uniform sampled data. In the algorithm by Shirley and Tuchman [ST90] tetrahedral cells of an unstructured grid are projected onto the image plane and the resulting triangles are blended appropriately. One drawback of this algorithm is that the cells need to be sorted according to their distance to the viewer's position. While the early splatting approaches suffered from severe aliasing artifacts, several people have worked on improving the image quality obtained from splatting [MMC99, ZPvBG02]. Examples for GPU implementations of splatting are given by, e.g., Chen et al. [CRZP04], and Neophytou and Müller [NM05].

In the object-based *texture-slicing volume rendering* the volume is sliced by a number of planes which are rendered using the texture hardware [KW03, XZC05], see also Figure 2.4, left. The volume rendering integral is approximated by blending the slices back-to-front in order to determine the final color for each fragment. This approach is very simple to implement and can be very fast (real-time volume visualization of up to 512^3 data points on consumer hardware is possible). Still based on trilinear interpolation aliasing artifacts are not avoidable. *Shear warp* [Wes89, LL94] is closely related to 2D texture-slicing, see Figure 2.4, right. Here, the volume is sheared such that viewing rays become perpendicular to one of the major axes of the data set and rows of voxels become aligned to rows of pixels in an intermediate image. This intermediate, but distorted picture is then warped to obtain the undistorted final picture.

The most flexible technique for volume rendering which also allows for the best visual quality is *ray casting* [Lev90, KW03, HLRSR09], see Figure 2.3, right. Rays emanating from the viewer's position are sent into the scene and we either integrate the equations of light transport along the rays for full volume rendering or find the first valid intersec-

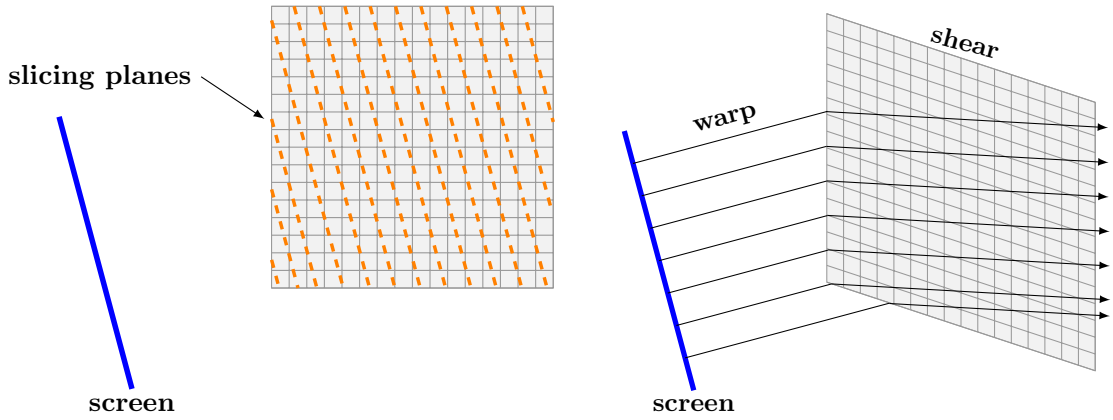

Figure 2.4: *Left*: texture slicing methods for volume rendering. The grid is sliced back-to-front by a number of planes parallel to the viewing plane. *Right*: shear-warp is closely related to two-dimensional texture slicing. Here, the volume itself is sheared such that viewing rays are perpendicular to one of the major axes of the data set.

tion with the zero contour for isosurfacing. Several optimizations to this basic principle have been proposed, for instance *empty space skipping*, *early ray termination*, *deferred shading*, adaptive sampling, and sophisticated caching systems for large data sets, e.g., *bricking* [PSL*98, HSS*05, MRH08]. Due to the simplicity of trilinear interpolation a basic GPU implementation for volume ray casting is straightforward to implement. Thus, most approaches are based on this local spline model and therefore trade visual quality in favor of rendering speed. Gradients can be precomputed at the grid points at the cost of increased memory and bandwidth consumption. Alternatively, the gradients are computed on-the-fly using central differences which is an expensive operation. Either way, the obtained gradients are not smooth and visual artifacts arise. An efficient implementation for interactive isosurface visualization with higher order filtering has been given by Hadwiger et al. [HS05, HSS*05]. These C^2-continuous splines lead to polynomials of total degree nine for which no exact root finding algorithms exist. Therefore, ray-surface intersections have to be found by interval refinement techniques. Due to the tensor-product structure of these splines evaluation of gradients require additional evaluations at the intersection point. Furthermore, data stencils are large (the $4 \times 4 \times 4$-neighborhood of the centering data value is used) and important features might get lost resulting from the large support of the filter kernels.

A different technique is *frequency domain volume rendering* (FDVR) [TL93]. FDVR is based on the *Fourier projection slice theorem* which states that extracting a slice perpendicular to the image plane and including the origin from the Fourier transformed volume data corresponds to the image obtained from integrating rays perpendicular to the image plane in the spatial domain. Although computational complexity is reduced from $\mathcal{O}(n^3)$ (where n is the number of data points in each spatial direction) to $\mathcal{O}(n^2 \log(n))$, FDVR can only be used with parallel projections. Further, occlusion effects are difficult to incorporate into FDVR with the consequence that the images resemble those obtained

from X-Rays. A GPU approach of FDVR is discussed in [VKG04].

In this chapter we discuss the most widespread reconstruction and visualization methods for volume data. Since our main contribution in this field is the development of efficient visualization methods for smooth trivariate splines, i.e., splines defined on uniform tetrahedral partitions of the volumetric domain, we give an overview of this related work in the following section.

2.3 GPU Visualization of Trivariate Splines

In this thesis we concentrate on trivariate splines defined w.r.t. tetrahedral partitions of the volumetric domain for high-quality visualization of volume data. Pure CPU implementations for quadratic super splines have been previously developed for isosurfacing [RZNS03, RZNS04a], explicit isosurface reconstruction by piecewise quadratic polynomials [NRZ07], scattered data approximation [RZNS04b], and for direct volume rendering [SZH*05] using shear-warp. This work is concerned with developing the first efficient GPU kernels for interactive reconstruction and visualization of isosurfaces based on trivariate splines. The uniformity of the underlying partition, the small data stencils and low polynomial degree of these quasi-interpolating splines and the availability of repeated averaging type procedures for fast on-the-fly computation of spline coefficients correspond with the general requirements of GPUs in order to exploit the provided parallelism.

In this thesis we develop and evaluate two different algorithms for interactive visualization of isosurfaces based on trivariate splines. The first approach can be categorized as a hybrid cell projection / ray casting technique where the bounding geometry of each polynomial piece (i.e., a tetrahedron) is first projected onto the screen. Ray-surface intersection tests are then performed for each of the resulting fragments. GPU-implementations based on this basic projection principle are described in the literature on visualization of algebraic surfaces from trivariate polynomials [Rei05, LB06], and for the special cases for piecewise quadratic polynomials in three variables [WMK04, SGS06, SWBG06], with several applications, e.g, visualization of ball and stick models of protein data using smooth ellipsoids. Using the B-form of a polynomial Loop and Blinn [LB06] exploit the idea of testing for intersections needed for ray casting in screen space. Since, according to their method, the projection step has to be done on the CPU only a small number of tetrahedra can be processed in this way. While the approach by Sigg et al. [SWBG06] works well for a set of (special) polynomials not being related by continuity and smoothness conditions, Stoll et al. [SGS06] give a GPU implementation for quadratic polynomials and particularly apply their approach to quadratic super splines. However, still this is a *pure polynomial* GPU approach which requires full information, i.e., the explicit geometry of each tetrahedron together with all the spline coefficients needs to be transferred to the GPU. Obviously, this leads to extreme redundancies. The work by Kloetzli et al. [KOR08] also considers splines on tetrahedral partitions and is thus closely related to ours. The authors construct C^0-continuous cubic splines on arbitrary uniform tetrahedral partitions by a least squares approximation of the given volume data and

render isosurfaces using a modification of the approach by Loop and Blinn [LB06] which is restricted to orthogonal projections. The spline structure is not fully exploited in this approach as well, which results in a high memory demand and only small or subsampled data sets can be rendered with interactive frame rates.

In contrast to this previous works our GPU implementations (involving the additional and complex cases of quadratic and cubic C^1-splines) show that more sophisticated algorithms are needed *to deal with splines*. More precisely, we have to take the complex structure of trivariate splines into consideration and organize the data streams determining the splines and their bounding geometry appropriately. In connection with improved culling techniques this results in an essential improvement of frame rates, and less memory usage, see Section 6.1.

While the hybrid cell projection / ray casting approach works well for medium-sized data sets and low performance GPUs our tests have shown that the bottleneck both in terms of frame rates as well as memory usage lies in the sheer amount (several millions) of tetrahedra which have to be processed in each frame. Note that this is also a problem of GPU algorithms based on refinement methods [SSS06b, BS05], or the display of subdivision surfaces [BS02, SJP05, LS08], where triangle processing becomes a limiting factor when the screen-space projections of the triangles are near pixel resolution (*micro polygons*). Another problem is high overdraw, i.e., many fragments fall onto the same pixel and costly operations need to be done for occluded fragments. Thus, our work has motivated a pure image based and *geometry free* volume ray casting approach for trivariate splines without the need to project each tetrahedron individually, see Section 6.2. The regular structure of the tetrahedral partitions allows for a very efficient traversal of rays throughout the volume where, unlike for irregular grids [WKME03], the needed intersections with the faces of the tetrahedra can be found very quickly. Further, we incorporate an efficient empty space skipping into the ray traversal based on octrees which leads to real-time and high-quality visualizations of large real-world data sets.

Besides pure visualization GPUs are nowadays also used for reconstruction of geometry from measured data. A variety of algorithms for hardware-based triangle mesh generation from volume data exists, i.e., marching cubes [DZTS06, TSD07, NVI08a] and marching tetrahedra [KW05]. These methods usually make use of parallel stream compaction to remove empty cells from processing and thus improve performance and memory usage of the generated meshes. To do that, one can use, e.g., *parallel prefix scans* [Ble90, HSO07] on the GPU. We apply similar concepts to our smooth splines in a preprocess prior to visualization in order to identify empty regions which can be excluded from costly ray-patch intersection tests. Because of the high computing power and low prices of current GPUs they are applied to many non-graphics related problems, a field which is known as *general purpose GPU* (GPGPU) [LHK*04]. This development and increasing flexibility of GPU architectures gave rise to a number of frameworks for GPU parallel programming beyond the graphics pipeline, one of the most mature and popular being NVIDIA's *CUDA* framework [NVI08a]. We show in Chapter 6 how we can benefit from CUDA in order to improve the performance of reconstruction and visualization algorithms based on trilinear splines.

Chapter 3

Bernstein-Bézier Techniques for Multivariate Polynomials

In this chapter, we present the fundamentals of the Bernstein-Bézier form (B-form) for curve and surface representations. We lay the theoretical foundations needed for a deeper understanding of the mathematical concepts used in this thesis, and we also present the tools needed to deal with B-form polynomials in a practical setting, for instance, its evaluation and derivative calculation.

We first give an overview of the univariate case, i.e., curves of arbitrary degree in B-form. This topic has already been extensively studied in the vast field of literature on Computer Aided Geometric Design (CAGD), e.g., [dB87, HL93, Far02]. Thus, we only recall the most important properties of curves in B-form here. The remainder of this chapter is dedicated to bivariate and trivariate polynomials in B-form, i.e., Bézier triangles and Bézier tetrahedra. Bézier triangles have been discovered by Paul de Casteljau in the 1960's as a generalization of univariate Bézier curves as a more natural surface representation than tensor products. The extension of the B-form to the three-dimensional simplex are the Bézier tetrahedra. The associated spline spaces, i.e., bivariate splines defined on triangulations in the plane, and trivariate splines on tetrahedral partitions of a three-dimensional domain, respectively, are discussed in Chapter 4.

The bivariate B-form naturally extends to the trivariate setting. Thus, for simplicity of notation we mainly derive the concepts of multivariate polynomials in B-form for the bivariate case, and just give the corresponding results for trivariate polynomials. Further aspects of bivariate polynomials in B-form are discussed in, e.g., [BF83, Far86, Sei89, PBP02, Far02, LS07]. The trivariate case is also briefly treated in [Far02, LS07].

3.1 Univariate Bernstein-Polynomials and Bézier Curves

According to the *Approximation Theorem of Weierstraß*, for any continuous function f in one variable on an interval $[a, b]$ we can find a polynomial which approximates $f(x)$, $x \in [a, b]$ with arbitrary precision. More precisely, for an $\epsilon > 0$, there exists a polynomial

$$p(x) := \sum_{i=0}^{q} a_i x^i \tag{3.1}$$

of degree q with $||f(x) - p(x)||_\infty < \epsilon$, $x \in [a, b]$. Besides the monomial form (3.1), several choices of polynomial bases are possible. For example, we could express p in

terms of Hermite or Lagrange basis polynomials or we could use the Newton basis. Each basis has different properties in terms of numerical stability, efficiency of evaluation, ease of geometric interpretation of the coefficients, etc. We use the *Bernstein-Bézier form* (B-form) of p,

$$p(x) := \sum_{i+j=q} b_{ij} B_{ij}(x), \tag{3.2}$$

where the b_{ij} are the *Bernstein coefficients* and the $q+1$ univariate *Bernstein basis polynomials* of degree q are given by

$$B_{ij}(x) := \frac{q!}{i!j!} \phi_0(x)^i \phi_1(x)^j.$$

Here, the $\phi_\mu := \phi_\mu(x)$, $\mu = 0, 1$, are linear polynomials determined by $\phi_0(x) := \frac{b-x}{b-a}$ and $\phi_1(x) := \frac{x-a}{b-a} = 1 - \phi_0(x)$, thus $\phi_0(x) + \phi_1(x) = 1$. The Bernstein coefficients b_{ij} are associated with the $q+1$ *Greville abscissae* or *Bézier points* $\xi_{ij} := \frac{ia+jb}{q}$, $i + j = q$. The Bernstein polynomials sum to one,

$$\sum_{i+j=q} B_{ij}(x) = 1, \quad \text{all } x \in \mathbb{R}, \tag{3.3}$$

which follows from the binomial expansion

$$1 = (\phi_0 + \phi_1)^q = \sum_{i+j=q} \frac{q!}{i!j!} \phi_0^i \phi_1^j,$$

and for any $x \in [a, b]$, we have $B_{ij}(x) \geq 0$. We can express the derivative w.r.t. ϕ_μ, $\mu = 0, 1$ of a Bernstein polynomial of degree q in terms of a Bernstein polynomial of degree $q-1$, for instance

$$\frac{\partial B_{ij}}{\partial \phi_0} = q \cdot \left[\frac{(q-1)!}{(i-1)!j!} \phi_0^{i-1} \phi_1 \right] = q B_{i-1,j}.$$

Using the chain rule, we see that the derivatives w.r.t. x are given as

$$\begin{aligned} B'_{ij}(x) &= \frac{q!}{i!j!} \cdot \left[\frac{-i}{b-a} \phi_0^{i-1}(x) \phi_1^j(x) + \frac{j}{b-a} \phi_0^i(x) \phi_1^{j-1}(x) \right] \\ &= \frac{q}{b-a} \cdot \left[B_{i,j-1}(x) - B_{i-1,j}(x) \right], \end{aligned} \tag{3.4}$$

where we use the convention that Bernstein polynomials with a negative subscript are considered as zero.

If we plot the *graph* of $p(x)$ for $x \in [a, b]$, we get the corresponding Bézier curve. For a cubic example, see Figure 3.1. The *control points* (ξ_{ij}, b_{ij}) of the curve have a geometric interpretation: since $\phi_0(a) = 1, \phi_0(b) = 0$, and $\phi_1(a) = 0, \phi_1(b) = 1$, the Bézier curve interpolates the Bernstein coefficients associated with a and b. Considering that $B_{ij}(x) = 0$, for $x = a$ and $i \neq q$, and $x = b$, $j \neq q$, respectively, and Equation (3.4), differentiating (3.2) in the endpoints of the interval simplifies to

$$p'(a) = \frac{q}{b-a} \cdot [b_{q-1,1} - b_{q0}] \quad \text{and} \quad p'(b) = \frac{q}{b-a} \cdot [b_{0q} - b_{1,q-1}].$$

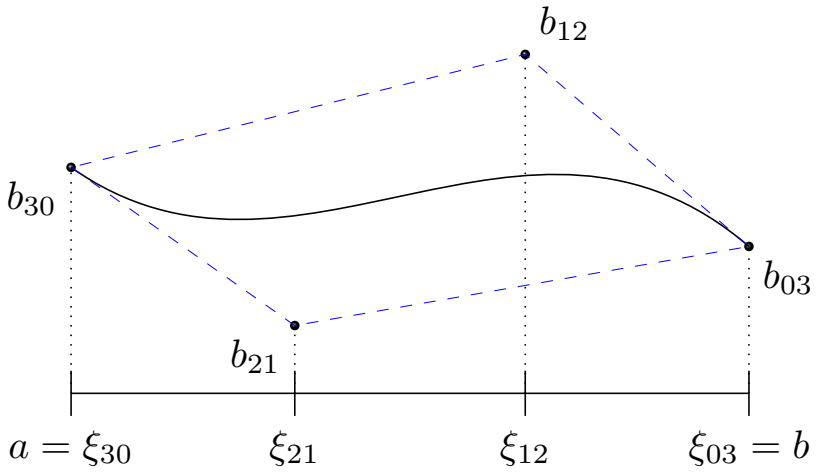

Figure 3.1: A cubic Bézier curve on an interval $[a, b]$ within its control polygon (dashed blue lines).

This means that the Bézier curve is tangential in a to the line determined by the first and second coefficient b_{q0} and $b_{q-1,1}$, with a similar interpretation for the tangent in b. Further, the Bézier curve is bounded by its *control polygon*, i.e., the convex hull of the points (ξ_{ij}, b_{ij}), which follows from Equation (3.3) and $\phi_0, \phi_1 \geq 0$, for all $x \in [a, b]$.

We can evaluate a Bézier curve numerically stable using the *de Casteljau algorithm*. The theory of Bézier curves can be easily extended to *univariate splines*, i.e., piecewise polynomials in one variable joining smoothly at the transitions. We close the discussion on univariate Bézier curves at this point and refer the interested reader to the vast amount of literature available on this topic, e.g., [PBP02, Far02]. Instead, in the following we directly examine the extension of the univariate theory to the less-known bivariate and trivariate settings for surface representations w.r.t. triangles and tetrahedra, respectively.

3.2 The Space of Bivariate and Trivariate Polynomials

In the following, let

$$\mathcal{P}_q := \operatorname{span}\{x^i y^j : 0 \leq i + j \leq q\}$$

be the $\binom{q+2}{2}$-dimensional space of bivariate polynomials of total degree q. For example, the quadratic polynomials in two variables,

$$p(x, y) = a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}, \quad a_{ij} \in \mathbb{R},$$

define the well-known class of conic sections in $\mathbb{R}^2$, i.e., ellipses, parabolas and hyperbolas.

Analogously, we consider the $\binom{q+3}{3}$-dimensional space of trivariate polynomials of

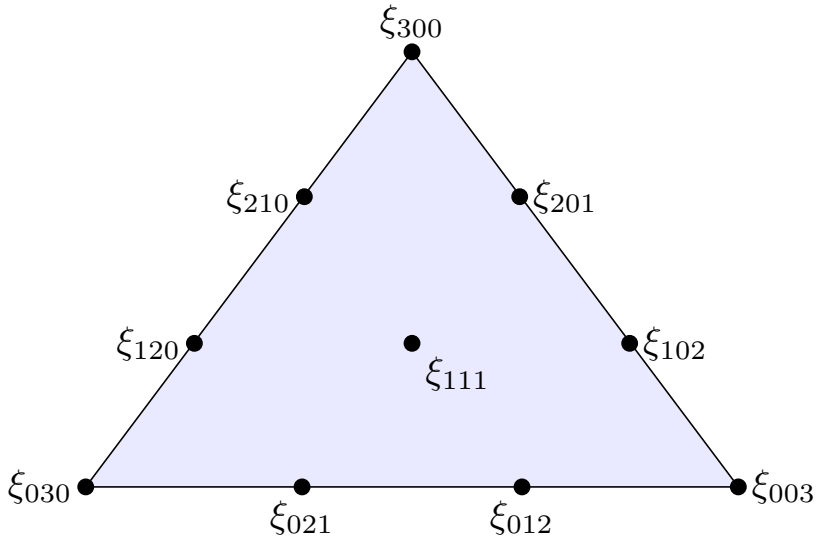

Figure 3.2: The domain points ξ_{ijk}, $i+j+k=q$, on a triangle for $q=3$.

degree q,

$$\mathcal{P}_q := \operatorname{span}\{x^i y^j z^k : 0 \leq i+j+k \leq q\}.$$

As a popular example, the quadratic polynomials of the form

$$p(x,y,z) = a_{200}x^2 + a_{110}xy + a_{101}xz + a_{020}y^2 + a_{011}yz + a_{002}z^2 + a_{100}x + a_{010}y + a_{001}z + a_{000},$$

with $a_{ijk} \in \mathbb{R}$, describe 3D surfaces known as quadrics, such as ellipsoids, paraboloids or hyperboloids.

We proceed with a simple construction of a set of bivariate Lagrange polynomials, which interpolates the values associated with a certain set of points arranged on a triangle, before we define a more suitable basis for our purposes in Section 3.3.

3.2.1 Lagrange Interpolation on Triangles

Let $\triangle = [v_0, v_1, v_2]$ be a non-degenerate triangle with vertices $v_\mu = (x_\mu, y_\mu)^\mathsf{T} \in \mathbb{R}^2$, $\mu = 0,1,2$ and a set of $\binom{q+2}{2}$ points

$$\xi_{ijk} := \frac{iv_0 + jv_1 + kv_2}{q}, \quad i+j+k=q.$$

In the following, we call the ξ_{ijk} *domain points* of $\triangle$, and define the set of domain points associated with $\triangle$ as $\mathcal{D}_q(\triangle) := \{\xi_{ijk}\}_{i+j+k=q}$. The domain points for $q=3$ are shown in Figure 3.2.

We now construct a bivariate polynomial of degree q, which interpolates given values z_{ijk} at the domain points ξ_{ijk} of $\triangle$, see [LS07]. To do this, we first define linear polynomials a_μ, b_ν, c_κ such that

$$\begin{aligned} a_\mu(v) = 0 &\text{ is the line passing through the points } \xi_{\mu jk} \text{ with } \mu + j + k = q, \\ b_\nu(v) = 0 &\text{ is the line passing through the points } \xi_{i\nu k} \text{ with } i + \nu + k = q, \text{ and} \\ c_\kappa(v) = 0 &\text{ is the line passing through the points } \xi_{ij\kappa} \text{ with } i + j + \kappa = q, \end{aligned}$$

with $v := (x, y)^\mathsf{T} \in \mathbb{R}^2$. For example,

$$a_0(v) = (x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1)$$

describes the line passing through the points ξ_{0q0} and ξ_{00q}, i.e., the edge opposite to ξ_{q00}. The bivariate *Lagrange polynomials* of degree q

$$p_{ijk}(v) := \prod_{\mu=0}^{i-1} \frac{a_\mu(v)}{a_\mu(\xi_{ijk})} \prod_{\nu=0}^{j-1} \frac{b_\nu(v)}{a_\nu(\xi_{ijk})} \prod_{\kappa=0}^{k-1} \frac{c_\kappa(v)}{c_\kappa(\xi_{ijk})},$$

where we use the convention that each product term with a negative superscript is set to one, satisfy

$$p_{ijk}(\xi_{\mu,\nu,\kappa}) = \begin{cases} 1, & (i, j, k) = (\mu, \nu, \kappa), \\ 0, & (i, j, k) \neq (\mu, \nu, \kappa). \end{cases}$$

By construction of the p_{ijk}, it immediately follows that the unique polynomial of degree q that interpolates the z_{ijk} at the domain points is

$$p := \sum_{i+j+k=q} z_{ijk} p_{ijk}.$$

It can be shown (see [LS07]) that p approximates smooth functions with an error bound K solely dependent on the smallest angle in $\triangle$ and the polynomial degree q.

3.3 The Multivariate Bernstein Basis

In the following, we want to derive a basis of $\mathcal{P}_q$ which has several useful properties. This basis is affine invariant, and stable and efficient techniques for (simultaneous) evaluation of the polynomials and its derivatives are available. Further, smoothness conditions between neighboring polynomials can be defined using simple averaging formulae, which simplifies the structural analysis of smooth spline spaces. We first introduce the concept of *barycentric coordinates* with respect to the simplexes in $\mathbb{R}^d$, $d = 2, 3$, i.e., triangles and tetrahedra. Next, we discuss Bernstein basis polynomials, which are based on barycentric coordinates. The remainder of this chapter deals with the B-form of bivariate and trivariate polynomials, along with the central tool for the evaluation of polynomials and its derivatives, the de Casteljau algorithm.

3.3.1 Barycentric Coordinates

Let $\triangle$ be a non-degenerate triangle in the plane with vertices $v_\mu := (x_\mu, y_\mu)^\mathsf{T} \in \mathbb{R}^2$, $\mu = 0, 1, 2$. For any point $v = (x, y)^\mathsf{T} \in \mathbb{R}^2$ exists a set of three scalars $\phi_\mu, \mu = 0, 1, 2$, with

$$\phi_0 + \phi_1 + \phi_2 = 1,$$

such that v can be uniquely described by the affine combination

$$v = \phi_0 v_0 + \phi_1 v_1 + \phi_2 v_2.$$

This relation can be also written in matrix form as

$$\begin{pmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3.5}$$

Let $s = v_1 - v_0$ and $t = v_2 - v_0$, the triangle's area $A_\triangle$ is one half of the area of the parallelogram spanned by s and t. This area is determined by the length of the cross product of the vectors s and t embedded in $\mathbb{R}^3$:

$$A_\triangle = \frac{1}{2} ||\mathbf{s} \times \mathbf{t}||_2,$$

where $|| \cdot ||_2$ is the usual 2-norm and $\mathbf{s} = (s_x, s_y, 0)^\mathsf{T}$, $\mathbf{t} = (t_x, t_y, 0)^\mathsf{T}$. Note that the length of the cross product corresponds to the determinant $|M_\triangle|$, where $M_\triangle$ is the matrix occurring in Equation (3.5):

$$A_\triangle = \frac{1}{2} |M_\triangle|.$$

From *Cramer's Rule* it follows that the barycentric coordinate ϕ_0 of v w.r.t. $\triangle$ is given by

$$\phi_0(v) = \begin{vmatrix} x & x_1 & x_2 \\ y & y_1 & y_2 \\ 1 & 1 & 1 \end{vmatrix} / |M_\triangle|, \tag{3.6}$$

with analogous expressions for ϕ_1 and ϕ_2. By expanding (3.6),

$$\phi_0(v) = \frac{x y_1 + x_2 y + x_1 y_2 - x_2 y_1 - x_1 y - x y_2}{x_1 y_2 + x_0 y_1 + x_2 y_0 - x_1 y_0 - x_2 y_1 - x_0 y_2}, \tag{3.7}$$

we see that the ϕ_μ are linear polynomials with $\phi_\mu(v_\mu) = 1$ and $\phi_\mu(v_\nu) = 0, \mu \neq \nu$, i.e.,

$$\phi_\mu(v_\nu) = \delta_{\mu,\nu}, \quad \mu, \nu = 0, 1, 2,$$

where $\delta_{\mu,\nu}$ is *Kronecker*'s symbol.

Let $\triangle_0 = [v, v_1, v_2]$, $\triangle_1 = [v_0, v, v_2]$ and $\triangle_2 = [v_0, v_1, v]$, we have the geometric interpretation of barycentric coordinates as ratios of triangle areas,

$$\phi_\mu = \frac{A_{\triangle_\mu}}{A_\triangle}, \quad \mu = 0, 1, 2,$$

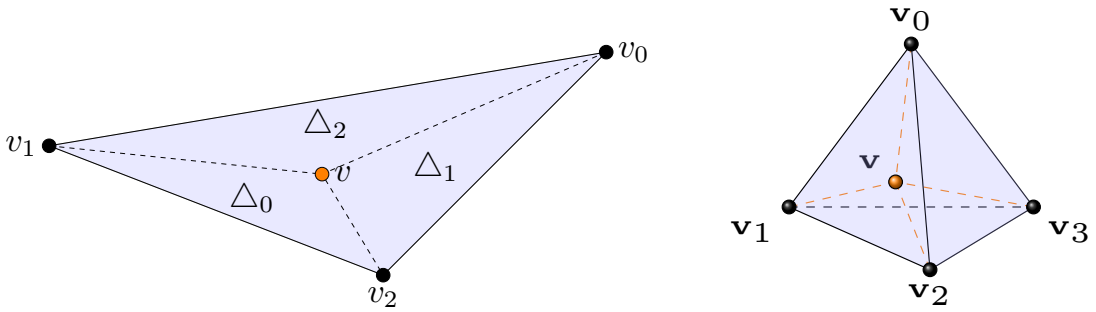

Figure 3.3: *Left*: The barycentric coordinates of a point $v \in \mathbb{R}^2$ w.r.t. a triangle $\triangle = [v_0, v_1, v_2]$ are given as ratios of the areas of the subtriangles $\triangle_0 = [v, v_1, v_2], \triangle_1, \triangle_2$ with the area of $\triangle$. *Right*: The barycentrics of a point $\mathbf{v} \in \mathbb{R}^3$ w.r.t. a tetrahedron $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ are analogously given as ratios of the volumes of the subtetrahedra $T_0 = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}], T_1, T_2, T_3$, with the volume of T.

see Figure 3.3, left. Further, a point v is *inside* $\triangle$ iff all barycentric coordinates of v w.r.t. $\triangle$ are positive. Note that for points on the edges of $\triangle$, at least one of the corresponding barycentric coordinate is always zero.

Barycentric coordinates are *affine invariant*, i.e., for any affine map Φ of the form

$$\Phi(v) := Av + w,$$

where Av is a linear map and $w \in \mathbb{R}^2$, we have

$$\Phi(v) = A\sum_{i=0}^{2}(\phi_i v_i) + w \overset{*}{=} \sum_{i=0}^{2}(\phi_i A v_i) + \sum_{i=0}^{2}\phi_i w = \sum_{i=0}^{2}\phi_i(Av_i + w) = \sum_{i=0}^{2}\phi_i \Phi(v_i),$$

with the second step following from $\phi_0 + \phi_1 + \phi_2 = 1$. Thus, the barycentric coordinates of a point v w.r.t. the original triangle $\triangle$ are the same as the barycentric coordinates of the transformed point $\Phi(v)$ w.r.t. the transformed triangle $\Phi(\triangle)$. Affine maps include translations and rotations as special case.

The concept of barycentric coordinates can be easily extended to three dimensions. The 3D simplex describing a volume is a tetrahedron T with vertices $[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$, where $\mathbf{v}_\mu = (x_\mu, y_\mu, z_\mu)^\mathsf{T} \in \mathbb{R}^3$, $\mu = 0, 1, 2, 3$. For any point $\mathbf{v} = (x, y, z)^\mathsf{T} \in \mathbb{R}^3$ exists a set of four scalars ϕ_μ, $\mu = 0, 1, 2, 3$, with

$$\phi_0 + \phi_1 + \phi_2 + \phi_3 = 1,$$

such that $\mathbf{v}$ is uniquely determined by the affine combination

$$\mathbf{v} = \phi_0\mathbf{v}_0 + \phi_1\mathbf{v}_1 + \phi_2\mathbf{v}_2 + \phi_3\mathbf{v}_3.$$

Analogous to Equation (3.5), this can be written as

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \phi_3 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{3.8}$$

The determinant of the matrix M_T occuring in the above equation is connected to the volume V_T of T via $6 \cdot V_T = |M_T|$. For a point $\mathbf{v}$, we have the barycentrics $\phi_\mu, \mu = 0, 1, 2, 3$, as the ratios of the volumes of the subtetrahedra $T_0 = [\mathbf{v}, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3], T_1 = [\mathbf{v}_0, \mathbf{v}, \mathbf{v}_2, \mathbf{v}_3], T_2 = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}, \mathbf{v}_3]$ and $T_3 = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}]$, and T:

$$\phi_\mu(\mathbf{v}) = \frac{V_{T_\mu}}{V_T}, \quad \mu = 0, 1, 2, 3,$$

see Figure 3.3, right. Again, the ϕ_μ are linear polynomials with $\phi_\mu(\mathbf{v}_\nu) = \delta_{\mu,\nu}$, $\mu, \nu = 0, 1, 2, 3$, and $\mathbf{v}$ is *inside* of T iff all its barycentrics are positive. On each of the faces of T, at least one barycentric coordinate is zero, and on each edge of T, at least two barycentrics are zero.

Derivatives of Barycentric Coordinates

From Equation (3.7) it follows immediately that the derivatives

$$\frac{\partial \phi_\mu}{\partial x} \quad \text{and} \quad \frac{\partial \phi_\mu}{\partial y}, \quad \mu = 0, 1, 2,$$

are constant scalars characterized by the relative positions of the v_μ, i.e., the shape and orientation of the triangle. For example, $\frac{\partial \phi_0}{\partial x} = (y_1 - y_2)/|M_\triangle|$. We have similar equations for the 3D simplex, where the derivatives

$$\frac{\partial \phi_\mu}{\partial x}, \quad \frac{\partial \phi_\mu}{\partial y} \quad \text{and} \quad \frac{\partial \phi_\mu}{\partial z}, \quad \mu = 0, 1, 2, 3$$

are also constant fractions solely dependent on the shape and orientation of the associated tetrahedron.

Often, we need the inverse mappings of Equation (3.5), $\phi(v) = M_\triangle^{-1} \cdot v$, and Equation (3.8), $\phi(\mathbf{v}) = M_T^{-1} \cdot \mathbf{v}$. For practical purposes, it is interesting to notice that the rows of $M_\triangle^{-1}$ and M_T^{-1} correspond to the line and plane equations of the edges of $\triangle$ and the faces of T, respectively, and are connected to the derivatives of the barycentrics. For example, the line of the triangle edge opposite to vertex v_μ, is given by

$$l_\mu(x, y) = \frac{\partial \phi_\mu}{\partial x} \cdot x + \frac{\partial \phi_\mu}{\partial y} \cdot y + d_\mu$$

where $\frac{\partial \phi_\mu}{\partial x}, \frac{\partial \phi_\mu}{\partial y}, d_\mu$ are the components of the μth row of $M_\triangle^{-1}$.

Directional Coordinates

As shown in Section 3.3.1, the triple ϕ_μ, $\mu = 0, 1, 2$, with $\sum_\mu \phi_\mu = 1$, describes a point in $\mathbb{R}^2$ w.r.t. a triangle $\triangle$. Given two points a and b in $\mathbb{R}^2$, we gain a useful description of the difference vector $u = a - b$ using *directional coordinates* α_μ, which are given as the difference of the associated barycentric coordinates:

$$\alpha_\mu(u) := \phi_\mu(a) - \phi_\mu(b), \quad \mu = 0, 1, 2.$$

Note that the directional coordinates of a vector sum to 0. Assume $\phi_\mu(v)$, $\mu = 0, 1, 2$, are the barycentric coordinates of a point v and $\alpha_\mu(u)$, $\mu = 0, 1, 2$ are the directional coordinates of a vector u, then the barycentric coordinates of a point $v + t \cdot u$, $t \in \mathbb{R}$, are given as

$$\phi_\mu(v + t \cdot u) = \phi_\mu(v) + t \cdot \alpha_\mu(u), \quad \mu = 0, 1, 2. \tag{3.9}$$

Differentiating Equation (3.7) and summing terms gives

$$\frac{\partial \phi_0}{\partial x} + \frac{\partial \phi_1}{\partial x} + \frac{\partial \phi_2}{\partial x} = 0, \quad \text{and} \quad \frac{\partial \phi_0}{\partial y} + \frac{\partial \phi_1}{\partial y} + \frac{\partial \phi_2}{\partial y} = 0,$$

showing that the derivatives of barycentric coordinates are directional coordinates.

We can analogously define directional coordinates for 3D simplexes, where the difference vector $\mathbf{u} = \mathbf{a} - \mathbf{b}$ of two points $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ can be described in terms of directional coordinates as

$$\alpha_\mu(\mathbf{u}) := \phi_\mu(\mathbf{a}) - \phi_\mu(\mathbf{b}), \quad \mu = 0, 1, 2, 3,$$

with $\sum_\mu \alpha_\mu = 0$.

3.3.2 Bernstein Polynomials

Similar to the univariate case, we can define *bivariate Bernstein basis polynomials* of degree q w.r.t. a triangle $\triangle$ as

$$B^q_{ijk}(v) := \frac{q!}{i!j!k!} \phi_0(v)^i \phi_1(v)^j \phi_2(v)^k, \quad i + j + k = q,$$

and the *trivariate Bernstein basis polynomials* of degree q w.r.t. a tetrahedron T are given as

$$B^q_{ijk\ell}(\mathbf{v}) := \frac{q!}{i!j!k!\ell} \phi_0(\mathbf{v})^i \phi_1(\mathbf{v})^j \phi_2(\mathbf{v})^k \phi_3(\mathbf{v})^\ell, \quad i + j + k + \ell = q.$$

We usually omit the superscript since it follows from the context. Since the ϕ_μ are linear polynomials (see Section 3.3.1), the B_{ijk} and $B_{ijk\ell}$ are polynomials of total degree q. The basis polynomials form a partition of unity, e.g., for the bivariate case we have

$$\sum_{i+j+k=q} B_{ijk}(v) = 1, \quad \text{all } v \in \mathbb{R}^2, \tag{3.10}$$

which follows from the trinomial expansion

$$1 = (\phi_0 + \phi_1 + \phi_2)^q = \sum_{i+j+k=q} \frac{q!}{i!j!k!} \phi_0^i \phi_1^j \phi_2^k.$$

Combining this with the fact that each $\phi_\mu(v) \geq 0$ if $v \in \triangle$, we have

$$0 \leq B_{ijk}(v) \leq 1, \quad \text{all } v \in \triangle,$$

with an equivalent conclusion for the trivariate case. The derivative w.r.t. a barycentric coordinate ϕ_μ of a Bernstein basis polynomial of degree q can be written in terms of a basis polynomial of degree $q-1$, e.g.,

$$\frac{\partial B_{ijk}}{\partial \phi_0} = q \cdot \left[\frac{(q-1)!}{(i-1)!j!k!} \phi_0^{i-1} \phi_1^j \phi_2^k \right] = q \cdot B_{i-1,j,k}^{q-1}, \tag{3.11}$$

where Bernstein polynomials with negative subscripts are considered zero and with similar expressions for the remaining cases.

There are exactly $n_\triangle := \binom{q+2}{2}$ bivariate and $n_T := \binom{q+3}{3}$ trivariate Bernstein polynomials, which is equal to the dimensions of the bivariate and trivariate polynomial spaces $\mathcal{P}_q$, respectively. It can be shown (see [LS07]) that the set of bivariate and trivariate Bernstein polynomials form a basis for the space of bivariate and trivariate polynomials $\mathcal{P}_q$, respectively.

In the following, we imply a lexicographical order based on the multi-indices of the basis polynomials and domain points. For example, the order for the cubic bivariate basis polynomials is

$$B_{300}, \quad B_{210}, B_{201}, \quad B_{120}, B_{111}, B_{102}, \quad B_{030}, B_{021}, B_{012}, B_{003}.$$

This allows to unambiguously define an invertible mapping from an integral index $\iota \in [0, n_{\{\triangle,T\}} - 1]$ to the corresponding multi-index i, j, k or i, j, k, ℓ, respectively.

Directional Derivatives of Bernstein Basis Polynomials

Let f be a differentiable function, and u is a vector in $\mathbb{R}^2$. We now define the *directional derivative* of f at a point v w.r.t. u as

$$D_u f(v) := \frac{d}{dt} f(v + tu)|_{t=0}.$$

To derive the directional derivative of a bivariate Bernstein basis polynomial B_{ijk}, we use the connection between a vector u in $\mathbb{R}^2$ and its corresponding directional coordinates $(\alpha_0, \alpha_1, \alpha_2)$ and Equation (3.9) (see Section 3.3.1):

$$B_{ijk}^q(v + tu) = \frac{q!}{i!j!k!} \cdot [\phi_0(v) + t\alpha_0(u)]^i [\phi_1(v) + t\alpha_1(u)]^j [\phi_2(v) + t\alpha_2(u)]^k.$$

Differentiating w.r.t. t and evaluating at $t = 0$ gives

$$D_u B_{ijk}^q(v) = \frac{q!}{i!j!k!} \cdot [i\alpha_0 \phi_0^{i-1} \phi_1^j \phi_2^k + j\alpha_1 \phi_0^i \phi_1^{j-1} \phi_2^k + k\alpha_2 \phi_0^i \phi_1^j \phi_2^{k-1}].$$

Rearranging terms, we arrive at the directional derivative of B_{ijk}^q w.r.t. u:

$$D_u B_{ijk}^q(v) = q \cdot \left[\alpha_0 B_{i-1,j,k}^{q-1}(v) + \alpha_1 B_{i,j-1,k}^{q-1}(v) + \alpha_2 B_{i,j,k-1}^{q-1}(v) \right], \tag{3.12}$$

where we use the convention that terms with negative subscripts are considered zero. The directional derivatives of a trivariate basis polynomial $B_{ijk\ell}^q$ are defined in a similar fashion.

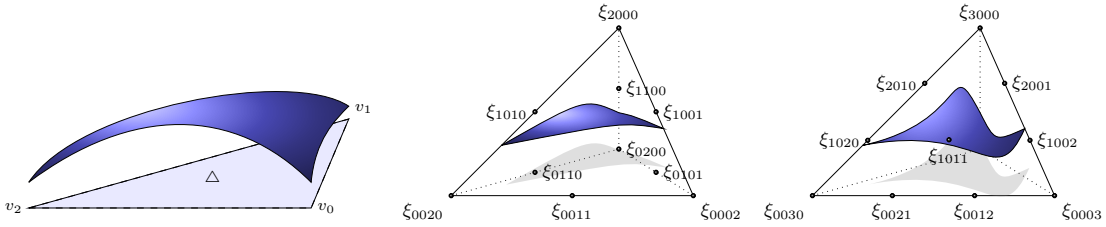

Figure 3.4: Examples of surfaces defined by functional polynomials in B-form. *Left*: Surface of a bivariate polynomial, where the $p(v)$ can be interpreted as height values over the domain triangle $\triangle$. *Middle*: (simple) zero contour of a trivariate quadratic polynomial defined w.r.t. a tetrahedron T along with the domain points on T. *Right*: (simple) zero contour of a trivariate cubic polynomial. Only the domain points on the front-most triangle are shown.

Maxima of Bernstein Basis Polynomials

We show that each Bernstein basis function B_{ijk} w.r.t a triangle $\triangle = [v_0, v_1, v_2]$ has a unique maximum at the domain point $\xi_{ijk} := (iv_0 + jv_1 + kv_2)/q$, see [LS07]:

$$B_{ijk}(v) < B_{ijk}(\xi_{ijk}), \quad v \in \triangle, v \neq \xi_{ijk} \text{ and } i + j + k = q.$$

First we consider the case where $i, j, k > 0$, i.e., the domain point ξ_{ijk} lies in the interior of $\triangle$ and B_{ijk} vanishes on all three edges of $\triangle$. In order to identify an extremum, we have to calculate the derivatives of B^q_{ijk} in two different directions. Using the linearly independent vectors $v_0 - v_1$ and $v_0 - v_2$, with associated directional coordinates $u_0 = (1, -1, 0)$ and $u_1 = (1, 0, -1)$, respectively, in Equation (3.12) gives

$$D_{v_0 - v_1} B_{ijk}(v) = B_{ijk}(v) \left(\frac{i}{\phi_0} - \frac{j}{\phi_1} \right),$$
$$D_{v_0 - v_2} B_{ijk}(v) = B_{ijk}(v) \left(\frac{i}{\phi_0} - \frac{k}{\phi_2} \right).$$

Since $B_{ijk}(v) \geq 0$ and $\phi_0 + \phi_1 + \phi_2 = 1$, we can see that the two expressions vanish simultaneously if and only if

$$i\phi_1 - j\phi_0 = 0,$$
$$i(1 - \phi_0 - \phi_1) - k\phi_0 = 0.$$

Substituting the first equation into the second gives us the unique solution $(\phi_0, \phi_1, \phi_2) = (i/q, j/q, k/q)$, which are the barycentric coordinates of ξ_{ijk}. It follows that B_{ijk} has a unique maximum at ξ_{ijk}. The proofs for the remaining choices of i, j, k are similar.

The maxima of the trivariate basis polynomials are also given at the domain points, which can be shown with a similar procedure.

3.4 The Bernstein-Bézier Form

Since the bivariate Bernstein polynomials span a basis of $\mathcal{P}_q$, we can write any bivariate polynomial p of degree q w.r.t. a triangle $\triangle$ in its *Bernstein-Bézier form* (B-form), i.e.,

$$p|_{\triangle} := \sum_{i+j+k=q} b_{ijk} B_{ijk},$$

where the *Bernstein coefficients* or *Bézier points* b_{ijk} of p are associated with the domain points ξ_{ijk} of $\triangle$. At vertex v_0, $B_{q00} = 1$ and the remaining basis polynomials are zero, analogously for v_1 and v_2. Thus, the B-form interpolates the values at the vertices, e.g., $p(v_0) = b_{q00}$. Polynomials in B-form are affine invariant, since they are defined w.r.t. barycentric coordinates of a simplex and the basis functions sum to one. Since the maxima of the basis functions are given at the domain points, see Section 3.3.2, it immediately follows that for a point $v \in \triangle$, $|p(v)| \leq \max(|b_{ijk}|)$.

A bivariate polynomial in B-form (a Bézier triangle) describes a smooth surface as the graph of a bivariate function on the domain triangle $\triangle$, where the points $P(v) \in \mathbb{R}^3$, $v = (x, y)^{\mathsf{T}} \in \triangle$, of the surface are given by

$$P(v) := \begin{pmatrix} v \\ p(v) \end{pmatrix} = \sum_{i+j+k=q} \begin{pmatrix} \xi_{ijk} \\ b_{ijk} \end{pmatrix} B_{ijk}(v),$$

see Figure 3.4, left.

If we allow the Bézier points to be vectors, then $\mathbf{p}(v) := \sum_{i+j+k=q} \mathbf{b}_{ijk} B_{ijk}(v)$, for $v \in \triangle$, describes a parametric surface, where the $\mathbf{b}_{ijk} \in \mathbb{R}^3$ form the control polygon of $\mathbf{p}$, see Figure 3.5.

We can analogously write any trivariate polynomial w.r.t. a tetrahedron T as

$$p|_T := \sum_{i+j+k+\ell=q} b_{ijk\ell} B_{ijk\ell},$$

where the $b_{ijk\ell}$ are associated with the domain points $\xi_{ijk\ell}$ of T. Trivariate polynomials in B-form, i.e., Bézier tetrahedra, implicitly describe a smooth surface as a zero contour by the set of points $\{\mathbf{v} = (x, y, z)^{\mathsf{T}} : \mathbf{v} \in T \text{ and } p(\mathbf{v}) = 0\}$. Examples of simple surfaces defined as zero contours of trivariate quadratic and cubic polynomials are shown in Figure 3.4, middle and right, respectively.

3.4.1 The de Casteljau Algorithm

From the definition of the Bernstein polynomials immediately follows the recurrence relation

$$B^q_{ijk} = \phi_0 B^{q-1}_{i-1,j,k} + \phi_1 B^{q-1}_{i,j-1,k} + \phi_2 B^{q-1}_{i,j,k-1},$$

where terms with negative subscripts are considered to be zero. The *de Casteljau algorithm* for evaluating a B-form polynomial $p(v) = \sum_{i+j+k=q} b_{ijk} B_{ijk}(v)$ is based on this

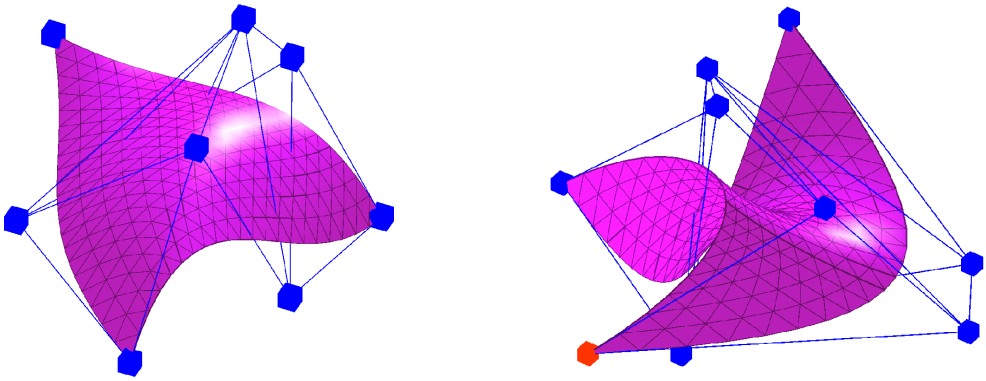

Figure 3.5: Two examples of bivariate, cubic parametric Bernstein-Bézier patches with their control polygons. The surface is visualized using a fine tessellation of the patches.

recursive definition of the Bernstein polynomials. Let $b_{ijk}^{[0]} := b_{ijk}$, $i+j+k=q$, then one de Casteljau step computes

$$b_{ijk}^{[r]} := \phi_0(v) b_{i+1,j,k}^{[r-1]} + \phi_1(v) b_{i,j+1,k}^{[r-1]} + \phi_2(v) b_{i,j,k+1}^{[r-1]}, \quad i+j+k = q-r. \tag{3.13}$$

We can rewrite any polynomial of degree q at v as a polynomial of degree $q-r$ using the intermediate coefficients $b_{ijk}^{[r]}$, $i+j+k=q-r$ as

$$p(v) = \sum_{i+j+k=q-r} b_{ijk}^{[r]} B_{ijk}^{q-r}(v), \quad 0 \le r \le q. \tag{3.14}$$

Further, the value of p at v is given as

$$p(v) = b_{000}^{[q]}. \tag{3.15}$$

Proof. The proof follows from induction on r. For $r=0$, Equation (3.14) is trivially true. Now, let us assume the equation holds for $r-1$. Using the recurrence relation for the B-polynomials of degree $q-(r-1) = q-r+1$, we have

$$\begin{aligned} p(v) &= \sum_{i+j+k=q-r+1} b_{ijk}^{[r-1]} B_{ijk}^{q-r+1}(v) \\ &= \sum_{i+j+k=q-r+1} b_{ijk}^{[r-1]} \cdot \left[\phi_0 B_{i-1,j,k}^{q-r}(v) + \phi_1 B_{i,j-1,k}^{q-r}(v) + \phi_2 B_{i,j,k-1}^{q-r}(v) \right]. \end{aligned}$$

If we split this sum into three individual sums and using an index shift, we have for the first sum

$$\sum_{i+j+k=q-r+1,\ i\ge 1} \phi_0 b_{ijk}^{[r-1]} B_{i-1jk}^{q-r}(v) = \sum_{i+j+k=q-r} \phi_0 b_{i+1,j,k}^{[r-1]} B_{ijk}^{q-r}(v).$$

Combining these sums again and using Equation (3.13) leads to (3.14). Finally, for $r=q$, (3.14) reduces to (3.15), since the only B-polynomial of degree zero is $B_{000}^0 = 1$. □

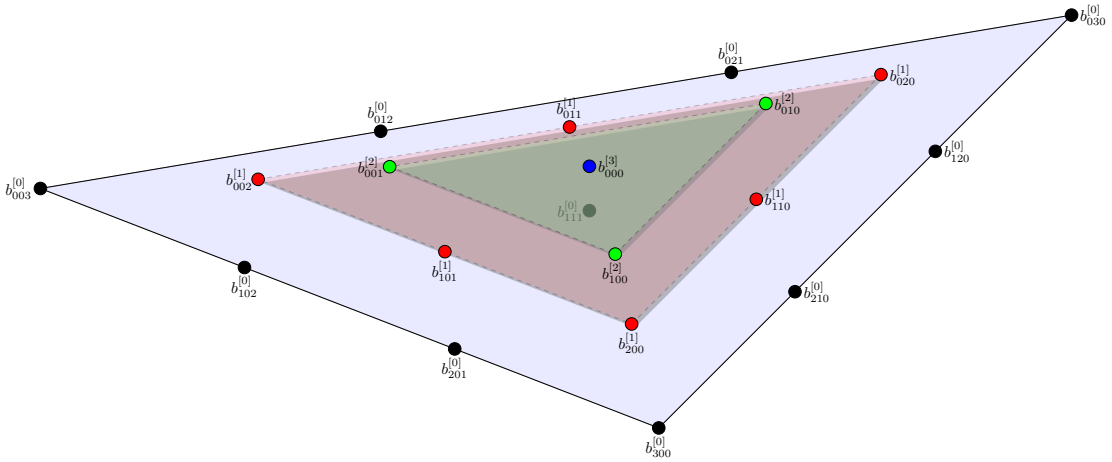

Figure 3.6: The de Casteljau pyramid for a cubic Bézier triangle. Any polynomial of degree q at a point v can be rewritten as a polynomial of degree $q - r$ using the intermediate coefficients occurring in the de Casteljau algorithm (shaded triangles).

The pyramid of coefficients $b^{[r]}_{ijk}$ occurring in the de Casteljau algorithm for a cubic bivariate polynomial is shown in Figure 3.6. The de Casteljau algorithm is numerical very stable for a point inside of $\triangle$, since in this case, the barycentric coordinates not only sum to one, but are also positive and thus each de Casteljau step consists of convex combinations of previously computed values.

The de Casteljau algorithm also delivers the derivatives of p in v w.r.t. the barycentric coordinates of v as a byproduct of the evaluation process. Let us consider the next to last step in the de Casteljau algorithm, where $q - r = 1$. Then, for example for ϕ_0 and using Equation (3.11), we have

$$\frac{\partial p(v)}{\partial \phi_0} = \sum_{i+j+k=1} b^{[q-1]}_{ijk} \frac{\partial B^1_{ijk}}{\partial \phi_0} = b^{[q-1]}_{100},$$

with similar results for the remaining cases. Differentiating again yields

$$\frac{\partial^2 p(v)}{\partial \phi_0^2} = \sum_{i+j+k=2} b^{[q-2]}_{ijk} \frac{\partial^2 B^2_{ijk}}{\partial \phi_0^2} = 2b^{[q-2]}_{200},$$

and the cross derivatives are given as $\partial^2 p(v)/\partial\phi_0\partial\phi_1 = 2b^{[q-2]}_{110}$, etc.

3.4.2 Directional Derivatives of B-Form Polynomials

We can extend the concept of directional derivatives from Section 3.3.2 to polynomials in B-form. First, we derive the derivative of p at v in the direction of $v_0 - v_1$, with associated directional coordinates $(1, -1, 0)$. Let us consider the next to last step of the

de Casteljau algorithm again, we have

$$D_{v_0-v_1}p(v) = \sum_{i+j+k=1} b_{ijk}^{[q-1]} D_{v_0-v_1} B_{ijk}^1(v).$$

Using Equation (3.12), and collecting non-zero terms, we get

$$D_{v_0-v_1}p(v) = q \cdot \left[b_{100}^{[q-1]} - b_{010}^{[q-1]} \right].$$

We can generalize this concept to arbitrary directions $u = a - b$ with associated directional coordinates $\alpha_\mu(u), \mu = 0, 1, 2$. The directional derivative of a polynomial p in B-Form at v in the direction of u is given by

$$D_u p(v) = q \cdot \left[\alpha_0 b_{100}^{[q-1]} + \alpha_1 b_{010}^{[q-1]} + \alpha_2 b_{001}^{[q-1]} \right],$$

where the $b_{ijk}^{[q-1]}$, $i + j + k = 1$, are the intermediate coefficients arising from $q - 1$ de Casteljau steps using v as argument. The evaluation of the directional derivative thus corresponds to $q-1$ de Casteljau steps using the barycentric coordinates of v, followed by one final de Casteljau step using the directional coordinates of u as input. This concept of mixed arguments in the de Casteljau algorithm is related to the *blossoming* principle, which gets a more in-depth treatment in Section 3.4.3.

Often, we need the derivatives of a polynomial in B-form in the direction of the Cartesian unit vectors, i.e., $x, y \in \mathbb{R}^2$ and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^3$, for example to perform illumination calculations for shading of a surface point. We have already shown in Section 3.3.1 that the derivatives of the barycentric coordinates are directional coordinates. In fact, the derivative of a bivariate polynomial in the direction of x is

$$D_x p(v) = q \cdot \left[\frac{\partial \phi_0}{\partial x} b_{100}^{[q-1]} + \frac{\partial \phi_1}{\partial x} b_{010}^{[q-1]} + \frac{\partial \phi_2}{\partial x} b_{001}^{[q-1]} \right], \tag{3.16}$$

with similar expressions for the remaining cases.

3.4.3 Blossoming

The blossoming principle is a generalization of the de Casteljau algorithm where the arguments may vary on the different levels [Ram87, Sei93]. Let p be a bivariate polynomial in B-form of degree q. If we use a_0 in the first level of the de Casteljau algorithm, a_1 on the next level, etc., then the *blossom* of p is denoted as

$$p\left[a_0, \ldots, a_{q-1}\right].$$

We often use the short notation $v^{<n>}$, which means that v is used n-times in the de Casteljau algorithm. It is easy to see that

$$p(v) = p\left[v^{<q>}\right].$$

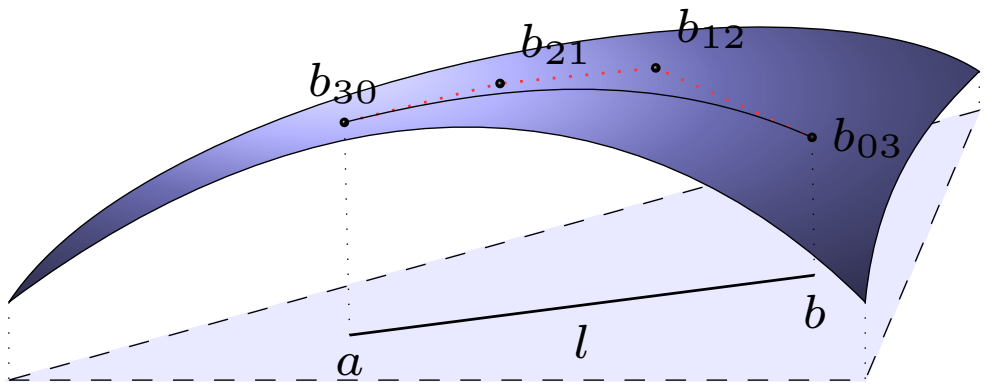

Figure 3.7: Bivariate blossoming for a cubic Bézier triangle. The straight line l on the domain triangle $\triangle$ is mapped to a cubic Bézier curve on the functional surface $P(v)$.

Further, the blossom is *symmetric*. Let π be an arbitrary permutation of the arguments $a_0, \ldots, a_{q-1}$, then

$$p\left[a_0, \ldots, a_{q-1}\right] = p\left[\pi(a_0, \ldots, a_{q-1})\right].$$

This means that the order in which the arguments appear is not relevant. Following [Far02, LS07], it is sufficient to show this exemplarily with a bivariate quadratic polynomial. Let a and b be two points in $\mathbb{R}^2$, then

$$\begin{aligned} p\left[a, b\right] = \; & \phi_0(b)(\phi_0(a)b_{200} + \phi_1(a)b_{110} + \phi_2(a)b_{101}) \\ & +\phi_1(b)(\phi_0(a)b_{110} + \phi_1(a)b_{020} + \phi_2(a)b_{011}) \\ & +\phi_2(b)(\phi_0(a)b_{101} + \phi_1(a)b_{011} + \phi_2(a)b_{002}) \end{aligned}$$

and

$$\begin{aligned} p\left[b, a\right] = \; & \phi_0(a)(\phi_0(b)b_{200} + \phi_1(b)b_{110} + \phi_2(b)b_{101}) \\ & +\phi_1(a)(\phi_0(b)b_{110} + \phi_1(b)b_{020} + \phi_2(b)b_{011}) \\ & +\phi_2(a)(\phi_0(b)b_{101} + \phi_1(b)b_{011} + \phi_2(b)b_{002}), \end{aligned}$$

thus

$$p\left[a, b\right] = p\left[b, a\right].$$

Because blossoms are evaluated using only linear combinations and the symmetry property, blossoms are *multi-affine*, i.e., for a scalar λ and an arbitrary level $0 \leq i \leq q-1$, it does not matter if we first form the barycentric combination of two (or more) points a, b, or if we combine the two individual blossoms:

$$p\left[\ldots, \lambda \cdot a + (1-\lambda) \cdot b, \ldots\right] = \lambda \cdot p\left[\ldots, a, \ldots\right] + (1-\lambda) \cdot p\left[\ldots, b, \ldots\right].$$

A useful property of the blossom is that it can be used to obtain the Bernstein coefficients b_{ijk} of a polynomial p in B-form using the triangle vertices as input. For example, if we use triangle vertex v_0 with barycentric coordinates $(1, 0, 0)$ as input for the first step, the intermediate result

$$p\left[v_0, \ldots\right]$$

consists of all Bernstein coefficients with the first index $i > 0$. Further,

$$b_{q00} = p\left[v_0^{<q>}\right],$$

and in general

$$b_{ijk} = p\left[v_0^{<i>}, v_1^{<j>}, v_2^{<k>}\right],$$

meaning that b_{ijk} is selected from p by the blossom using v_0 as input i times, j times v_1 and k times v_2, in arbitrary order. As a consequence, the Bernstein coefficients of the univariate boundary curve along the edge v_0, v_1 are given by the blossoms

$$b_{q-i,i} = p\left[v_0^{<q-i>}, v_1^{<i>}\right], \quad i = 0, \ldots q.$$

This can be generalized to arbitrary points a, b in the plane. Let l be the straight line in the domain defined by a, b, then the blossoms

$$p\left[a^{<q-i>}, b^{<i>}\right], \quad i = 0, \ldots q,$$

define a univariate Bernstein curve, which corresponds to the mapping of l onto the functional surface defined by p, see Figure 3.7.

In Section 3.4.2 we have shown that the directional derivative $D_u p(v)$ corresponds to $q - 1$ de Casteljau steps using v as input followed by one de Casteljau step using the directional coordinates of u. If we not only allow barycentric coordinates of a point as input to the de Casteljau algorithm, but also directional coordinates of a vector, we can rewrite the directional derivative as the blossom

$$D_u p(v) = q \cdot p\left[v^{<q-1>}, u\right]. \tag{3.17}$$

This principle can be extended to higher order derivatives. The rth directional derivative w.r.t. u in v is given by

$$D_u^r p(v) = \frac{q!}{(q-r)!} p\left[v^{<q-r>}, u^{<r>}\right].$$

Analogously, we obtain the mixed derivatives from the appropriate blossom. Let u_0 and u_1 be two vectors with associated directional coordinates, then

$$D_{u_0,u_1}^{r,s} p(v) = \frac{q!}{(q-r-s)!} p\left[v^{<q-r-s>}, u_0^{<r>}, u_1^{<s>}\right]$$

is the rth directional derivative of $p(v)$ w.r.t. u_0, followed by the sth directional derivative w.r.t. u_1.

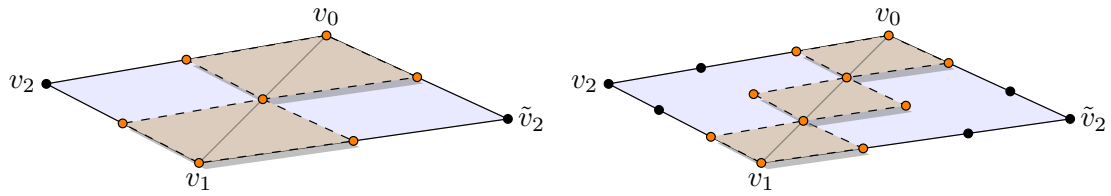

Figure 3.8: Bivariate C^1-smoothness conditions on two neighboring triangles $\triangle = [v_0, v_1, v_2]$ and $\tilde{\triangle} = [v_0, v_1, \tilde{v_2}]$. *Left*: coefficients involved in each of the C^1-smoothness conditions for a quadratic polynomial are shown as shaded trapezoids. *Right*: coefficients involved for C^1-smoothness in the cubic case.

3.4.4 Continuous Joins of Neighboring Polynomials

Until now, we have only considered single polynomials. In the following, we derive the conditions for *smooth joins* between two neighboring polynomials with common degree q. Let $\triangle = [v_0, v_1, v_2]$ and $\tilde{\triangle} = [v_0, v_1, \tilde{v}_2]$ be two neighboring triangles sharing the edge $e = \triangle \cap \tilde{\triangle} = [v_0, v_1]$. Let $p = \sum_{i+j+k=q} b_{ijk} B_{ijk}$ and $\tilde{p} = \sum_{i+j+k=q} \tilde{b}_{ijk} \tilde{B}_{ijk}$ be the two polynomials associated with $\triangle$ and $\tilde{\triangle}$, respectively. Any line in the domain is mapped onto a univariate curve on the surface, see Section 3.4.3. If the line crosses the common edge, we get a composite curve with one segment in each triangle. If all of the possible curves crossing e are C^r-continuous, then p and $\tilde{p}$ join with C^r-continuity. It immediately follows that for C^0-continuity, $b_{ij0} = \tilde{b}_{ij0}$, $i + j = q$. For C^1-continuity, we demand that for any direction u not parallel to e,

$$D_u p(v) = D_u \tilde{p}(v), \quad \text{all } v \in e.$$

Using Equation (3.17) and the symmetry of the blossoms we can write the above equation as

$$p\left[u, v^{<q-1>}\right] = \tilde{p}\left[u, v^{<q-1>}\right], \quad \text{all } v \in e.$$

Since the barycentric coordinates $\phi_\mu(v)$ w.r.t $\triangle$, and $\tilde{\phi}_\mu(v)$ w.r.t. $\tilde{\triangle}$, with $\mu = 0, 1, 2$, agree for each point v on the edge e, we have to consider the intermediate results from the first step of the blossoms using u as input:

$$p\left[u, ...\right] = \tilde{p}\left[u, ...\right].$$

It suffices to handle the case $u = \tilde{v}_2 - v_0$, since all directional derivatives of $p(v)$ and $\tilde{p}(v)$ w.r.t. the edge $e = [v_0, v_1]$ for all points v on e agree and derivatives in all other directions are linear combinations of D_u and $D_{v_1 - v_0}$. Let $\phi_\mu(\tilde{v}_2), \mu = 0, 1, 2$, be the barycentric coordinates of $\tilde{v}_2$ w.r.t. $\triangle$. The directional coordinates $\alpha_\mu(u), \mu = 0, 1, 2$, of $u = \tilde{v}_2 - v_0$ w.r.t. $\triangle$ are then given by $(\phi_0(\tilde{v}_2) - 1, \phi_1(\tilde{v}_2), \phi_2(\tilde{v}_2))$, and the directional coordinates $\tilde{\alpha}_\mu(u)$ of u w.r.t. $\tilde{\triangle}$ are $(-1, 0, 1)$. Now, it is easy to see that $p \cap \tilde{p}$ is C^1-continuous across e if and only if

$$\tilde{b}_{ij1} = b_{i+1,j,0}\, \phi_0(\tilde{v}_2) + b_{i,j+1,0}\, \phi_1(\tilde{v}_2) + b_{i,j,1}\, \phi_2(\tilde{v}_2), \quad i + j = q - 1. \tag{3.18}$$

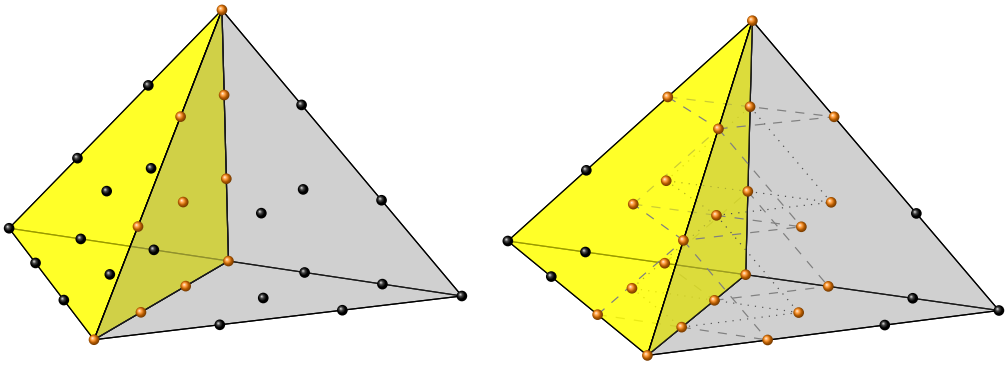

Figure 3.9: Trivariate smoothness conditions for two neighboring cubic polynomials sharing the face $[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$. *Left*: for C^0-continuity, the two neighboring tetrahedra share the coefficients on the common face (orange dots). *Right*: coefficients involved for C^1-smoothness across the common face of the tetrahedra are shown as orange dots.

Examples of the coefficients involved in C^1-smoothness for two quadratic and two cubic bivariate polynomials, respectively, are shown in Figure 3.8, left and right. While in general four coefficients are involved in each of these bivariate smoothness conditions, they degenerate to univariate conditions if one of the barycentric coordinates vanishes at $\tilde{v}_2$. This can happen for example if $v_2, v_1, \tilde{v}_2$ are collinear, since then $\phi_0(\tilde{v}_2) = 0$.

The above principle can be extended to higher order derivatives. In general, $p \cap \tilde{p}$ join with C^r-continuity if

$$\tilde{b}_{ijn} = \sum_{\nu+\mu+\kappa=n} b_{\nu+i,\mu+j,\kappa} B^n_{\nu\mu\kappa}(\tilde{v}_2), \quad i+j = q-n,\ n = 0, \ldots, r,$$

see [Far02, LS07].

Smoothness between two trivariate polynomials can be similarly defined. Let $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ and $\tilde{T} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \tilde{\mathbf{v}}_3]$ be two tetrahedra sharing a common face $F = T \cap \tilde{T} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$, and let $p(\mathbf{v}) = \sum_{i+j+k+\ell=q} b_{ijk\ell} B_{ijk\ell}$ and $\tilde{p}(\mathbf{v}) = \sum_{i+j+k+\ell=q} \tilde{b}_{ijk\ell} \tilde{B}_{ijk\ell}$ be the two polynomials associated with T and $\tilde{T}$, respectively. Then, $p \cap \tilde{p} \in C^0$ if

$$\tilde{b}_{ijk0} = b_{ijk0}, \quad i+j+k = q,$$

see Figure 3.9, left, for an example involving cubic Bézier tetrahedra, and $p \cap \tilde{p} \in C^1$ if

$$\tilde{b}_{ijk1} = b_{i+1,j,k,0}\, \phi_0(\tilde{\mathbf{v}}_3) + b_{i,j+1,k,0}\, \phi_1(\tilde{\mathbf{v}}_3) + b_{i,j,k+1,0}\, \phi_2(\tilde{\mathbf{v}}_3) + b_{i,j,k,1}\, \phi_3(\tilde{\mathbf{v}}_3), \quad i+j+k = q-1,$$

see Figure 3.9, right. There are five coefficients involved in each of these trivariate smoothness conditions. They degenerate to bivariate conditions if one of the barycentric coordinates vanishes at $\tilde{\mathbf{v}}_3$, for example when the faces $[\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_3]$ and $[\mathbf{v}_0, \mathbf{v}_1, \tilde{\mathbf{v}}_3]$ are coplanar. If $\mathbf{v}_3$ and $\tilde{\mathbf{v}}_3$ are collinear with either $\mathbf{v}_0, \mathbf{v}_1$ or $\mathbf{v}_2$, then two barycentric coordinates are zero and the above equations reduce to univariate conditions.

Chapter 4

Spaces of Smooth Splines

In this chapter, we briefly review the literature on bivariate and trivariate splines before we introduce the mathematical notation for a formal description of smooth spline spaces. Finally, we give two explicit examples of trivariate quasi-interpolating operators which have been proven useful for high-quality and real-time visualization of surfaces from volume data, see also Chapter 6.

4.1 Bivariate Splines for Data Approximation

Bivariate splines defined on triangulations in the plane are more flexible than tensor product splines since they can be defined both on uniform and on arbitrary triangulations. In Figure 4.1 we show examples of uniform and non-uniform triangulations of a planar domain. On the other hand, the spaces of smooth bivariate splines (let alone trivariate splines) are much more complex than tensor products and many problems of interest, such as the dimension (i.e., the number of coefficients which completely determine the spline) and approximation properties are not generally solved. Bivariate methods for data interpolation fall into two general classes: Classical finite element methods are based on *local Hermite interpolation* of high polynomial degree [Ž70, Sch89, DNZ01]. The spline is determined by interpolating function values and rth order derivatives at the vertices, and the cross-boundary derivatives at the edges of the triangulation. Splines of lower degree can be defined if all triangles are split leading to *macro-element methods* [BF80, CH90]. Two popular examples are the cubic C^1-continuous Clough-Tocher interpolant where each triangle is split into three subtriangles [CT65], and the quadratic C^1 Powell-Sabin interpolant which is based on splitting each triangle into six subtriangles [PS77]. Besides the high polynomial degree of finite elements and the high number of triangles for macro-elements a major problem of these methods is that the derivatives need to be estimated if they are not known. *Local Lagrange interpolation* by smooth splines is based on finding sets of interpolation points which uniquely determine the spline [NRSZ06]. No (cross-boundary) derivatives are needed but usually many of these interpolation points do not lie at the vertices of the triangulation and have to be generated in some way which might have negative effects on the quality of visualizations or the approximation properties. It is further difficult, often impossible for low polynomial degrees, to find suitable local interpolation sets for smooth splines because the degree of freedom is reduced by the continuity restrictions. This can be solved by using macro-elements, or by *priority* or *coloring* methods, where only certain subsets of tri-

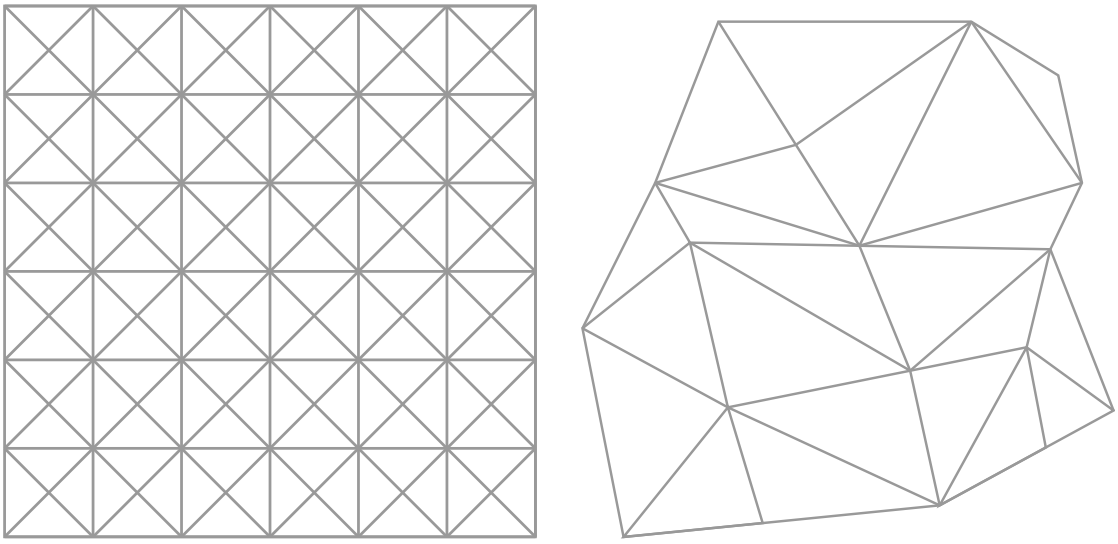

Figure 4.1: *Left*: a uniform triangulation of the rectangular domain. *Right*: an arbitrary non-uniform triangulation of a planar domain.

angles are split. For a survey of scattered data fitting methods using bivariate splines see [NZ00, Zei02].

A useful class for data approximation are *quasi-interpolating* operators where the piecewise polynomials are directly available from appropriate weightings of the data values [dBF73, MS07]. These methods often do not rely on triangle splits or prescribed derivatives and no additional interpolation points have to be generated [HZDS01, CJ05, SZ05]. The values at the grid points are approximated in a way such that a certain approximation order can be guaranteed, i.e., the error decreases by a known factor for increasing numbers of data points. On some uniform partitions, these splines have higher degree of freedom than those arising from Hermite or Lagrange interpolation which means that smoothness between neighboring polynomials can be achieved with a lower polynomial degree.

Bivariate splines in B-form have been successfully applied to several problems of interest in Computer Graphics. The curved PN triangles by Vlachos and Peters [VPBM01] can be used to approximate a curved parametric surface from a given triangle mesh using only local triangle data (i.e., normals at the vertices). The resulting piecewise cubic polynomials of this refinement method are C^1-continuous only at the vertices of the triangulation. Therefore, to achieve a smooth appearance of the surfaces, a separate quadratic varying normal model is applied. A similar approach based on quadratic Bézier triangles has been given by Bruijns [Bru98]. Zhao et al. approximate molecular surfaces with cubic splines which are C^1 only at the vertices and midpoints of the edges of the triangulation [ZXB07]. Theisel extends the marching cubes algorithm to calculate a G^1-continuous representation of isosurfaces using rational cubic triangular Bézier patches [The02]. Reis et al. apply triangular quartic C^1 and C^2-continuous quasi-interpolating splines in B-

form for smooth renderings of terrain data [RZHB*08, HBRZ09]. An approach for GPU visualization of two dimensional signed distance data using quadratic C^1-continuous quasi-interpolating splines has been given in [KTSZ08].

4.2 Trivariate Splines for Data Approximation

Trivariate splines are piecewise polynomials in three variables defined w.r.t. tetrahedral partitions of a volumetric domain. The polynomials are connected via smoothness conditions, see also Section 3.4.4. Worsey and Farin extended bivariate cubic C^1 Hermite interpolation to arbitrary n-dimensional simplexes [WF87, Sor06]. In the trivariate setting each tetrahedron needs to be split into twelve sub-tetrahedra. A piecewise quadratic Hermite interpolation scheme was then given by Worsey and Piper [WP88]. C^1-continuity is achieved by splitting each tetrahedron into twenty four sub-tetrahedra. Besides some new theoretical insights on the complex structure of trivariate spline spaces [HNR*04, HNZ06] several approximation and interpolation operators have been suggested recently. Local Lagrange interpolation of scattered data with C^1 quadratic and cubic trivariate splines has been given in [NSZ05, HNSZ08, HNSZ09], where (certain) tetrahedra are split into up to twenty four sub-tetrahedra. The cubic and quintic *A-Patches* by Bajaj et al., which are C^1 and C^2-continuous, respectively, can be used to construct smooth surfaces from the simplicial hull of a given surface triangulation [BCX95b, BCX95a]. These methods also rely on splits of certain tetrahedra of the simplicial hull to obtain global smoothness. Trivariate splines in B-form have been applied to several practical problems, e.g., the approximation of Navier-Stokes equations for 3D fluid simulation [AL04], the simulation of deformable models [WKG*11], or numerical solutions of general fourth-order differential equations [LW01]. While bivariate splines can be either used in a functional or parametric setting trivariate splines usually describe a surface implicitly. Implicit surfaces have higher degree of freedom and are closed under modeling operations such as union or intersection [VdFG98]. On the other hand, no direct representation of the surface exists and the zero contours are visualized indirectly, usually by Marching Cubes or ray casting.

We focus on quasi-interpolating operators defined w.r.t. uniform tetrahedral partitions for visualizations of volume data. Examples of commonly used tetrahedral partitions are shown in Figure 4.2. A major advantage of these operators is that the spline coefficients on each tetrahedron are immediately available from appropriate weightings of the data points in a small neighborhood of the centering data value. Schumaker and Sorokina [SS04] define quintic C^1-continuous local Lagrange and Hermite operators as well as a quasi-interpolating operator. They use the *type-4* partition (see Figure 4.2, left) where each data cube is split into five tetrahedra. Although the number of tetrahedra is relatively low the high polynomial degree makes this method less suitable for interactive GPU implementations. A good compromise between polynomial degree and number of tetrahedra is given by the quadratic super splines and cubic C^1-splines [NRSZ05, SZ07b] which are based on the *type-6* partition where each cube is split into twenty four congruent tetrahedra (see Figure 4.2, right). In our work, we focus on these quasi-interpolating

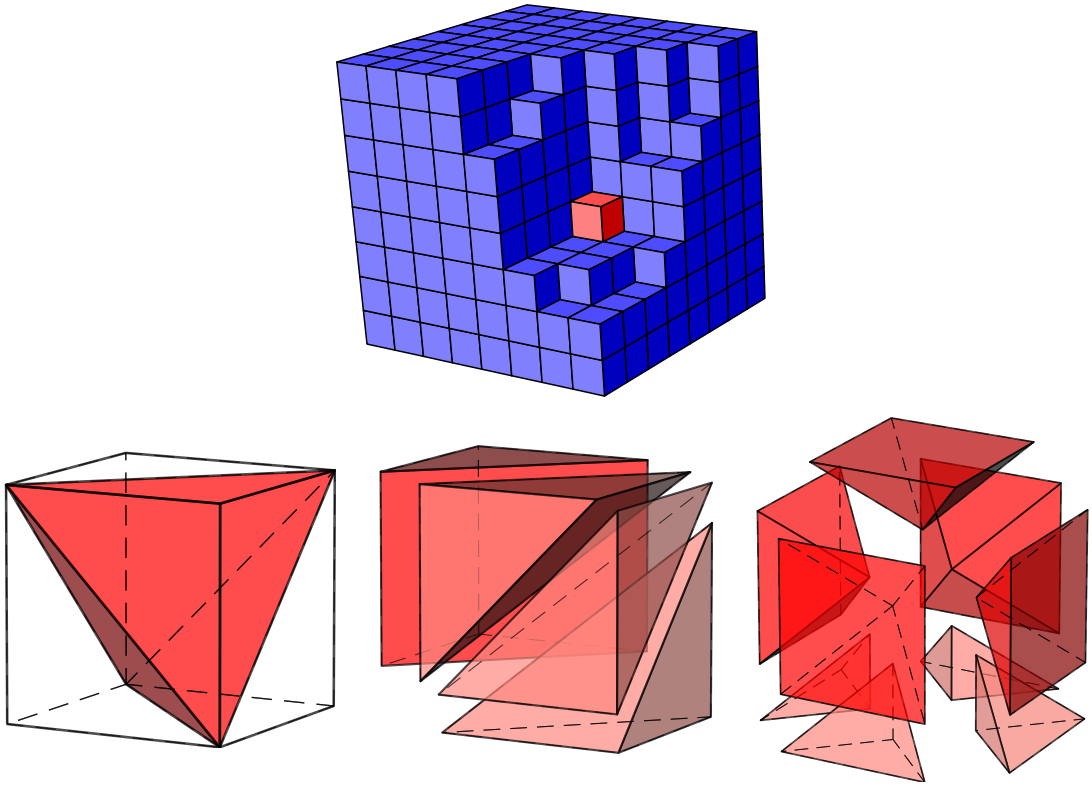

Figure 4.2: Three different uniform tetrahedral partitions of a cubic grid (top). *Left*: the type-4 partition is obtained by slicing each cube with four planes which results into five tetrahedra. *Middle*: in the Freudenthal partition each cube is first split into two prisms which are then further split into three tetrahedra each. *Right*: the type-6 partition is obtained by slicing each data cube with six planes resulting in twenty four congruent tetrahedra.

operators which have been proven suitable for real-time and high-quality visualizations of volume data, see Chapter 6. We further develop a new spline which is based on a more complex tetrahedral partition of truncated octahedra. This operator solves the problem of finding a C^1-continuous quadratic spline approximating data without the need to further subdivide any tetrahedra, see Chapter 5.

4.3 Smooth Spline Spaces

Let $\triangle$ be a triangular partition of the domain $\Omega \in \mathbb{R}^2$, and let r be an integral value with $0 \leq r \leq q$. We then define by

$$\mathcal{S}_q^r(\triangle) := \{s \in C^r(\Omega) : s|_{\triangle} \in \mathcal{P}_q, \text{ all } \triangle \in \triangle\}$$

the associated space of polynomial splines of degree q and smoothness r. Let $\mathcal{D}_q(\triangle) := \bigcup_{\triangle \in \triangle} \mathcal{D}_q(\triangle)$ be the union of the set of domain points associated with the triangles $\triangle$ of $\triangle$. It is easy to see that $s \in \mathcal{S}_q^0$ is a continuous spline uniquely defined by the coefficients $\{b_\xi :$

$\xi \in \mathcal{D}_q(\triangle)\}$. Therefore, the dimension of the space of quadratic continuous splines $S_2^0(\triangle)$ is equal to the cardinality of $\mathcal{D}_q(\triangle)$ and it follows that

$$\dim \mathcal{S}_2^0(\triangle) = V + E,$$

where V, E are the number of vertices and edges in $\triangle$, respectively. Let $\mathcal{M} \subseteq \mathcal{D}_q(\triangle)$, and $b_\xi \in \mathcal{M}$ be the associated B-coefficients, then we call $\mathcal{M}$ a *determining set* of $\mathcal{S}_q^r$ if all the remaining coefficients $b_\xi, \xi \notin \mathcal{M}$, are uniquely determined from the smoothness conditions given in Section 3.4.4. It is easy to see that $b_\xi = 0$, all $\xi \in \mathcal{M}$, implies $s = 0$.

For some applications, it is useful to define splines which have extended smoothness in certain points. For example, let $\mathcal{V} := \{v_0, \ldots, v_n\}$ be the set of vertices in $\triangle$, then

$$\mathcal{S}_q^{r,\tilde{r}}(\triangle) := \{s \in \mathcal{S}_q^r : s \in C^{\tilde{r}}(v), \text{ all } v \in \mathcal{V}\},$$

with $\tilde{r} > r$, denotes the associated space of *super splines* which are $C^{\tilde{r}}$-continuous in all vertices of $\triangle$, and C^r-continuous everywhere else. The PN triangles [VPBM01] by Vlachos and Peters are a popular example in Computer Graphics for a bivariate super spline $\mathcal{S}_3^{0,1}$, which is C^1-continuous in each vertex of $\triangle$, but only continuous across the edges of $\triangle$.

We can analogously define trivariate spline spaces w.r.t. tetrahedral partitions Δ of a volumetric domain $\Omega \in \mathbb{R}^3$ by

$$\mathcal{S}_q^r(\Delta) := \{s \in C^r(\Omega) : s|_T \in \mathcal{P}_q, \text{ all } T \in \Delta\},$$

with similar definitions of $\mathcal{D}_q(\Delta)$, super splines $\mathcal{S}_q^{r,\tilde{r}}$, and determining sets $\mathcal{M}$.

4.3.1 Quasi-Interpolating Splines

Quasi-interpolating operators are a very useful class of multivariate splines. We focus on trivariate quasi-interpolating operators for real-time reconstruction and high-quality visualization of volume data. Let Δ be a tetrahedral partition and $\Omega \in \mathbb{R}^3$ the domain covered by Δ. Now, let $f \in C(\Omega)$ be a continuous function and $\mathcal{N}$ is a subset of points in Ω (for example the grid points). We assume that the values of f are known at the points in $\mathcal{N}$. We now determine the Bézier coefficients $b_\xi, \xi \in \mathcal{D}_q(\Delta)$ of a continuous spline $s \in \mathcal{S}_q^0(\Delta)$ by

$$b_\xi = \sum_{\eta \in \mathcal{N}} \alpha_{\xi,\eta} f(\eta), \quad \xi \in \mathcal{D}_q(\Delta),$$

with $\alpha_{\xi,\eta} \in \mathbb{R}$ and $\sum_{\eta \in \mathcal{N}} \alpha_{\xi,\eta} = 1$, for $\xi \in \mathcal{D}_q(\Delta)$. This defines a linear map $\mathcal{Q} : C(\Omega) \to \mathcal{S}_q^0(\Delta)$, which has the important property that $\mathcal{Q}(1) = 1$. We call $\mathcal{Q}$ a *quasi-interpolating operator* for $\mathcal{S}_q^0(\Delta)$. For each tetrahedron T we further define $star^1(T) := T$ and recursively define $star^i(T)$, $i > 1$, as the union of $star^{i-1}(T)$ with the tetrahedra in Δ which have non-empty intersections with $star^{i-1}(T)$. We call $\mathcal{Q}$ *local* if there exists an $l \in \mathbb{N}$ such that for all $T \in \Delta$

$$a_{\xi,\eta} = 0, \quad \xi \in \mathcal{D}_q(T) \text{ and } \eta \notin star^l(T).$$

We can thus calculate all coefficients b_ξ for each polynomial $s_f := \mathcal{Q}(f)$ by the values of f in a (small) neighborhood $\Omega_T := star^l(T)$ of the tetrahedron T.

If further holds that the error on each T is bounded by a constant $C_T > 0$, i.e.,

$$||s_f||_T \leq C_T ||f||_{\Omega_T},$$

then $\mathcal{Q}$ is a *stable* operator. If the quasi-interpolating operator $\mathcal{Q}$ is local and stable, then the associated approximation method is also local and stable. Note that the constant C_T depends on f, the polynomial degree q as well as the smallest angle of the triangles in Δ. This means that the choice of tetrahedral partition Δ (the triangular partition for bivariate splines) has a direct effect on the approximation quality of the splines.

If we choose the weights $a_{\xi,\eta}$ appropriately such that the smoothness conditions (3.18) are satisfied simultaneously for all faces of Δ, then we can also define an approximation method w.r.t. smooth splines $\mathcal{S}_q^r(\Delta)$, $r < q$. In contrast to Hermite or Lagrange operators we have to verify that $\mathcal{Q}$ has the desired approximation properties, i.e., a certain approximation order. Roughly said, a local and stable quasi-interpolating operator $\mathcal{Q}$ inherits the approximation order of a polynomial space $\mathcal{P}_q$ if $\mathcal{Q}$ reproduces this space, i.e., $\mathcal{Q}(p) = p$ for all $p \in \mathcal{P}_q$. While it is almost trivial to construct continuous splines from given data it is known [Chu89, HNZ06, LS07] that incorporating appropriate smoothness conditions is a highly complex task for this type of splines. In particular, this holds true when the polynomial degree q is low, say $q \in \{2, 3\}$. On the other hand, it is also known [Nie00, WS01, BMDS02, RZNS04a, RZHB*08] that sophisticated rendering methods require smoothness conditions to be satisfied for the underlying trivariate models.

In the next section, we give two explicit examples of quasi-interpolating operators, namely trivariate $S_2^{0,1}$ and S_3^1 splines on type-6 tetrahedral partitions Δ_6. In Chapter 5 we further introduce a new S_2^1 spline defined on a tetrahedral partition of truncated octahedra.

4.4 Piecewise Quadratic and Cubic Approximation by Trivariate Splines

For $n \in \mathbb{N}$ let $\mathcal{V} := \{\mathbf{v}_{ijk} = (ih, jh, kh)^\mathsf{T} : i, j, k = 0, \ldots, n\}$ be the cubic grid of $(n+1)^3$ points with grid size $h = 1/n \in \mathbb{R}$. We define a cube partition $\square := \{Q : Q = Q_{ijk}\}$ of the domain Ω, where each $Q_{ijk} \in \square$ is centered at $\mathbf{v}_{ijk}$ and the vertices of Q_{ijk} are $(2i \pm 1, 2j \pm 1, 2k \pm 1)^\mathsf{T} \cdot h/2$, see Figure 4.2, top.

We consider trivariate splines on the *type-6 tetrahedral partition* Δ_6, where each Q is subdivided into twenty four congruent tetrahedra. This is done by connecting the vertices of Q_{ijk} with the center $\mathbf{v}_{ijk}$. Each of the resulting six pyramids is then further split into four tetrahedra, see Figure 4.2, bottom right. For each T we set $\mathbf{v}_0$ to the center of its cube Q, $\mathbf{v}_1$ to the centroid of one of the faces of Q, and $\mathbf{v}_2$ and $\mathbf{v}_3$ to the vertices of Q sharing a common edge.

Trivariate splines s w.r.t. Δ_6 are piecewise polynomials in three variables x, y, z of (total) degree q, which should be at least continuous, i.e. for all $T \in \Delta_6$, we have

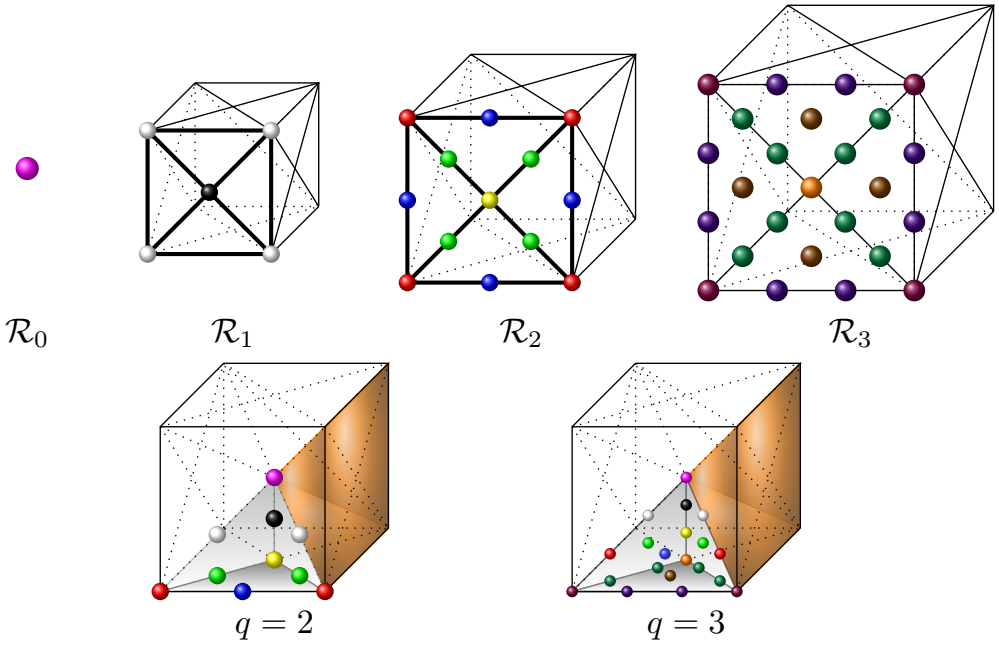

Figure 4.3: *Top*: the domain points in Q are organized in rings $\mathcal{R}_\nu,\ \nu = 0, \ldots, q$, around $\mathbf{v}_Q$. *Bottom left*: the 10 domain points of a quadratic polynomial w.r.t to a tetrahedron of Q . *Bottom right*: the 20 domain points of a cubic polynomial.

$p := s|_T \in \mathcal{P}_q := \text{span}\{x^i y^j z^k : \ i, j, k \geq 0,\ i + j + k \leq q\}$, and for any tetrahedra $T, \tilde{T} \in \Delta_6$ with $F = T \cap \tilde{T} \neq \emptyset$, we always assume that $p|_F = \tilde{p}|_F$.

We use the piecewise B-form of s, i.e.

$$s|_T = \sum_{i+j+k+\ell=q} b_{ijk\ell}\ B_{ijk\ell}, \quad \text{all } T \in \Delta_6, \tag{4.1}$$

see Section 3.4, where $B_{ijk\ell} \in \mathcal{P}_q$ are the Bernstein polynomials w.r.t. $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. The B-coefficients $b_{ijk\ell}$ are associated with the domain points $\xi_{ijk\ell} = (i\mathbf{v}_0 + j\mathbf{v}_1 + k\mathbf{v}_2 + \ell\mathbf{v}_3)/q$, and we let $\mathcal{D}_q(\Delta_6) := \{\xi_{ijk\ell} : \ T \in \Delta_6\}$ be the union of the sets of domain points associated with the tetrahedra of Δ_6. For each cube Q, the points from $Q \cap \mathcal{D}_q(\Delta_6)$ are organized in *rings* $\mathcal{R}_\nu,\ \nu = 0, \ldots, q$, around $\mathbf{v}_Q$ as shown in Figure 4.3, top.

Simple characterizing formulae describing smooth joins for neighboring polynomials exist for the piecewise B-form [Far86], see Section 3.4.4. A basic step in the structural analysis [HNR*04, NRSZ05, SZ07b] shows that C^1-smoothness for splines on Δ_6 requires simple formulae of the form

$$b_0 = (b_1 + b_2)/2 \ \text{ and } \ b_0 = (b_1 + b_2) - (b_3 + b_4)/2 \tag{4.2}$$

to be satisfied, where $b_\nu = b_\xi$ are certain B-coefficients associated with domain points ξ on the common triangular face $F = T \cap \tilde{T}$ and within distance 1 to F (i.e. $\xi = \xi_{ijk1}$). The second formula characterizes the smoothness of the neighboring polynomials if T and $\tilde{T}$

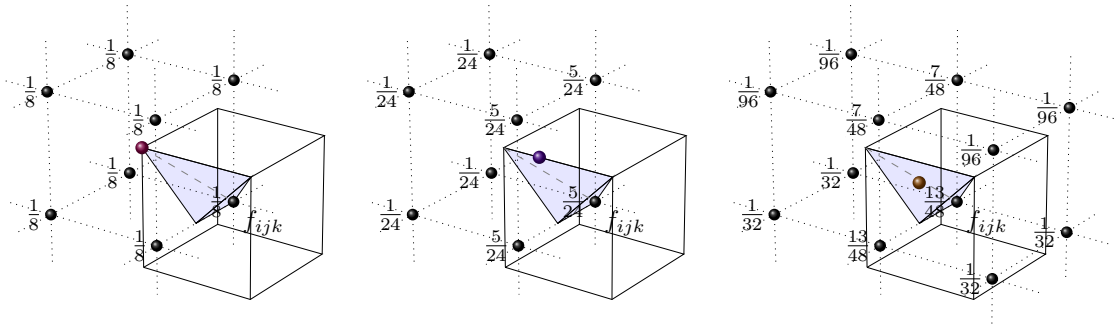

Figure 4.4: The masks for the coefficients associated with the domain points ξ_{0003} (left), ξ_{0021} (middle) and ξ_{0011} (right) for the cubic spline $s \in \mathcal{S}_3^1(\Delta_6)$. Colored dots correspond to Figure 4.3, black dots illustrate data values.

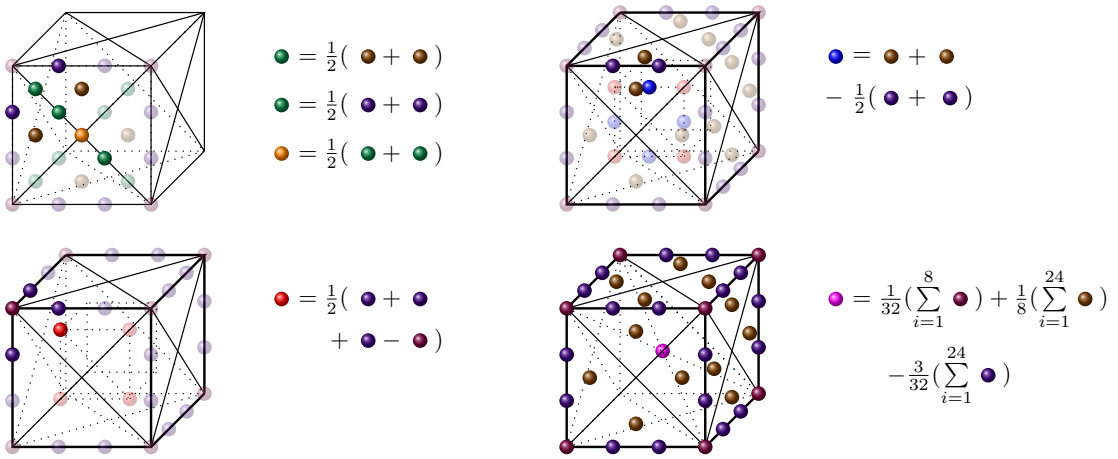

Figure 4.5: The remaining B-coefficients of a spline $s \in \mathcal{S}_3^1(\Delta_6)$ on the four rings $\mathcal{R}_v$ around $\mathbf{v}_Q$ can be obtained from repeated averagings of the coefficients in the determining set.

are contained in different pyramids of the same cube, while the first formula holds true in the remaining cases. Hence, Equation (4.2) shows that the smoothness of splines on Δ_6 is easily described when considering only two neighboring polynomials in B-form by means of averaging the involved B-coefficients. On the other hand, when modeling huge volume data sets many polynomial pieces on complete tetrahedral partitions Δ_6 have to be computed and the conditions in Equation (4.2) cannot be considered independently since they have to be satisfied simultaneously across all the interior triangular faces of Δ_6. This is the main reason for the highly complex structure of the spline spaces and has to be taken into account for any reconstruction method and the related visualization algorithms built on top of them.

Given *volume data*, i.e. scalar values $f_{ijk} \in \mathbb{R}$ associated with the grid positions $\mathbf{v}_{ijk}$, quasi-interpolating splines s on Δ_6 are directly determined by simply setting their B-

coefficients appropriately. More precisely, we consider the local representation of s, see Equation (4.1), and set its coefficients b_ξ, $\xi \in \mathcal{D}_q(\Delta_6)$, as certain weighted sums of some local portion of the data,

$$b_\xi = \sum_{i_0,j_0,k_0} \alpha_{i_0 j_0 k_0} \; f_{i+i_0,j+j_0,k+k_0}, \tag{4.3}$$

where $\alpha_{i_0 j_0 k_0}$ are fixed fractions weighting the values $f(\mathbf{v}_{i+i_0,j+j_0,k+k_0})$ close to $\mathbf{v}_{ijk}$. This approach leads to *real-time reconstructions* (for timings, see Section 6.1.3) since the B-coefficients of s are immediately available from the (typically huge) data without the need of any intermediate computations such as proper derivative estimation, matrix inversions, or usage of certain (locally supported) spanning splines.

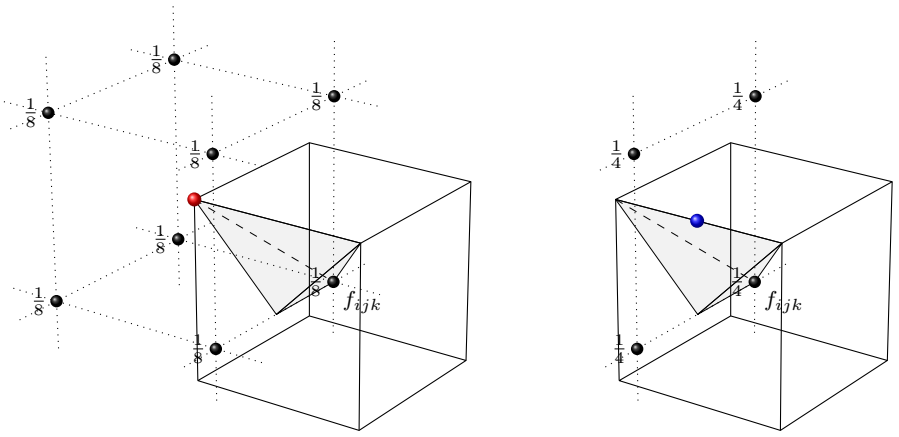

Figure 4.6: The masks for the coefficients associated with the domain points ξ_{0002} (left), and ξ_{0011} (right) for the quadratic super spline $s \in \mathcal{S}_2^{0,1}(\Delta_6)$. Colored dots correspond to Figure 4.3, black dots illustrate data values.

A main challenge of the above approach is often to find appropriate weights in Equation (4.3) and choices of i_0, j_0, k_0, such that the quasi-interpolants of low degree automatically satisfy smoothness conditions and some additional important properties (small data stencils, symmetry, guaranteed approximation order, automatic derivative approximation, stability of the operator). In our GPU implementations, we consider the symmetric quasi-interpolation schemes for quadratic [NRSZ05, RZNS04a] and cubic [SZ07b] splines on Δ_6. These methods are connected with certain determining sets $\mathcal{M} \subseteq \mathcal{D}_q(\Delta_6)$ such that the B-coefficients $b_\xi, \xi \in \mathcal{M}$, determine the spline by using Equations (4.2). The construction of the cubic quasi-interpolant is illustrated in Figure 4.4, where we show the masks for the coefficients $b_\xi \in \mathcal{M}$ obtained from averaging the data values in the 27-neighborhood of the centering data value. Here, we use a symmetric determining set $\mathcal{M}_Q$ for each $Q \in \square$, formed by the coefficients associated with the domain points $\xi_{00k\ell} \bigcup \xi_{0111}$, where $k + \ell = 3$. For a tetrahedron, ξ_{0030} and ξ_{0003} are vertices of Q, ξ_{0021} and ξ_{0012} are on the outer edges of Q, and ξ_{0111} corresponds to the centroid of the face $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$. The remaining B-coefficients follow from the smoothness conditions

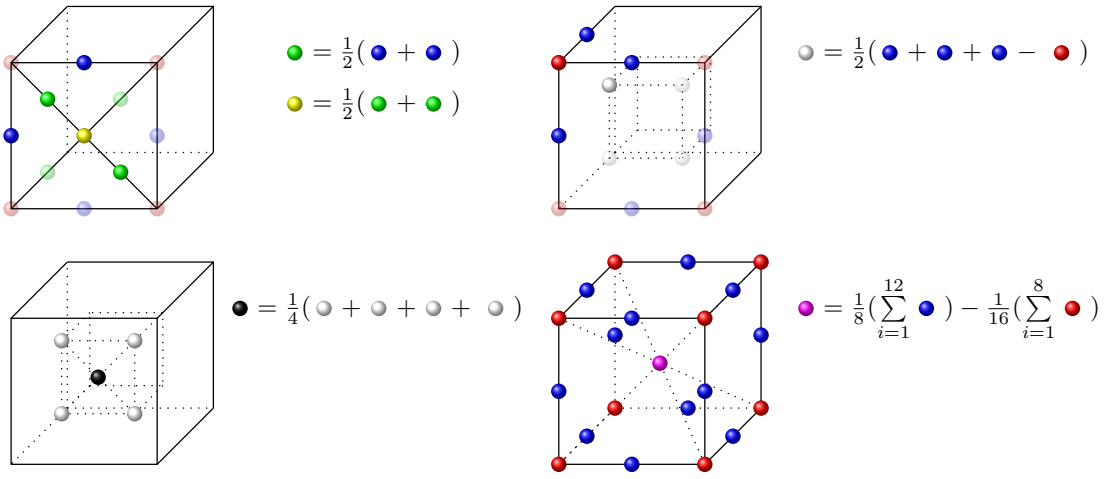

Figure 4.7: The remaining B-coefficients of a spline $s \in \mathcal{S}_2^{0,1}(\Delta_6)$ on the three rings $\mathcal{R}_v$ around $\mathbf{v}_Q$ can be obtained from repeated averagings of the coefficients in the determining set.

and are obtained by applying the rules shown exemplarily for a subset of coefficients in Figure 4.5. The remaining cases follow from symmetry and rotation. Explicit formulae for the B-coefficients can be found in [SZ07b].

The quadratic quasi-interpolant $s \in \mathcal{S}_2^{0,1}(\Delta_6)$ can be described similarly. The masks for the coefficients of the determining set $\mathcal{M} \subseteq \mathcal{D}_2(\Delta_6)$ are shown in Figure 4.6. The remaining coefficients are then obtained by the rules shown in Figure 4.7. The two quasi-interpolating splines differ in their smoothness properties: quadratic super splines are smooth on all the faces of the cubes Q in $\square$ while certain B-coefficients associated with domain points in the interior of Q are determined by averaging of smoothness conditions. As an example consider the black coefficient in Figure 4.7. The construction in [SZ07b] solves the problem of finding an overall C^1-continuous cubic spline $s \in \mathcal{S}_3^1(\Delta_6)$ approximating data locally without any tetrahedron subdivisions. Another difference is the number of B-coefficients within one cube which is 65 for quadratics, and 175 for cubics, respectively. On the other hand, both schemes are based on a small and symmetric data stencil using the 27-neighborhood of the centering data value, i.e., $i_0, j_0, k_0 \in \{-1, 0, 1\}$, see Figures 4.4 and 4.6. Tests with smooth functions (see also Section 5.5) confirmed that the quadratic and cubic quasi-interpolating splines as well as their first derivatives yield approximation order two, where, somewhat surprisingly, we observe that the constants are slightly better for cubics. The operator norm is an upper limit of the local change of the spline if one data value is varied. Since the weights for both operators are non-negative and sum to one, operator norms are also one for uniformly spaced data, which means that the spline does not change more than the data value. In case of non-uniformly spaced data, the operator norms are close to one.

Chapter 5

Quadratic C^1-Splines on Truncated Octahedral Partitions

In this chapter we describe a new approximating scheme for the smooth reconstruction of discrete data on volumetric grids. A local quasi-interpolation method for quadratic C^1-splines based on uniform tetrahedral partitions is used to achieve a globally smooth function. The Bernstein-Bézier coefficients of the piecewise polynomials are thereby directly determined by appropriate combinations of nearby data values, see also Section 4.3.1. We explicitly give a construction scheme for a family of quasi-interpolation operators and prove that the splines and their derivatives can provide an approximation order two for smooth functions. The optimal approximation of the derivatives and the simple averaging rules for the coefficients recommend this method for high quality visualization of volume data. Numerical tests confirm the approximation properties and show the efficient computation of the splines.

Trivariate C^1-splines on tetrahedral partitions, where the piecewise polynomials are given in their Bernstein-Bézier form are well suited for the purpose of volume approximation and visualization, since the Bernstein-Bézier techniques can be fully employed for the efficient computation and evaluation of the splines, see [Far86, dB87]. However, these are very complex spaces and only a few results are known up to date, see [LS07] and the references therein. Furthermore, C^1-splines with good approximation properties that solve Hermite- and Lagrange-Interpolation problems must have a relatively high degree (at least 5), see [Ž70, LM04, SS04], or make use of macro-elements, see [WF87, WP88, SW08], and [SSW09], where the tetrahedra of the partition are further subdivided, which results in complex partitions with a huge number of tetrahedra. Compared to that, quasi-interpolation (see [dBF73]) is a good approach which allows the usage of low degree splines on uniform partitions, while still some approximation properties can be guaranteed. The recent work of Sorokina and Zeilfelder [SZ07b] is most comparable to our approach. They avoid most of the above mentioned problems by constructing a C^1-smooth, cubic spline model based on a type-6 tetrahedral partition of the domain, see Section 4.4. The splines are built from appropriate averaging formulas in a symmetric and local neighborhood of the data values in a way such that the resulting splines approximate the values and the first derivatives of a sufficiently smooth function simultaneously with order two. Previous works (see [HNR*04, NRSZ05]) have shown that these results cannot be significantly improved. Particularly, the degree of the splines cannot be further decreased without the loss of either the approximation

properties or the C^1-smoothness of the resulting splines.

We describe the first C^1-smooth *quadratic* spline model on volumetric grids. Thereby, we employ a different uniform tetrahedral partition, which is obtained by subdividing each truncated octahedron of a truncated octahedral partition into 144 disjoint tetrahedra. This split is more complex than the 24-split of the cubes in the type-6 partition. On the other hand, the C^1-splines on truncated octahedral partitions possess significantly more degrees of freedom than C^1-splines on cubic partitions. For the approximation of the same volume data set, our spline method requires only approximately 50% more tetrahedra, while allowing for the lowest possible spline degree. The low degree of the polynomials results in a more efficient evaluation of the spline pieces, e.g., for intersecting rays with spline patches and the calculation of derivatives. We are able to describe an entire family of quasi-interpolating C^1-splines on truncated octahedral partitions and prove that for a specific spline the same approximation properties are achieved as for the cubic splines proposed in [SZ07b]. This means that our quasi-interpolating splines are able to approximate the derivatives of sufficiently smooth functions with optimal approximation order.

The remainder of this chapter is organized as follows. In Section 5.1, we recall important facts on trivariate splines. Then, in Section 5.2, we introduce truncated octahedral partitions and analyze the structure of C^1-splines on these partitions. In Section 5.3, we describe our quasi-interpolation scheme for the construction of a family of C^1-splines and prove some important results for certain quasi-interpolation operators. Section 5.4 contains the approximation properties of our method and in Section 5.5 we confirm these theoretical results with numerical tests. Finally, Section 5.6 concludes with comments on our future research and additional remarks.

5.1 Preliminaries

Let Δ be a tetrahedral partition of a set Ω in $\mathbb{R}^3$. In the following, we consider the space of quadratic C^1-splines on Δ, defined by

$$\mathcal{S}_2^1(\Delta) := \{s \in C^1(\Omega) : \ s|_T \in \mathcal{P}_2, \text{ for all } T \in \Delta\},$$

where $C^1(\Omega)$ is the set of continuously differentiable functions on Ω. We use the well known Bernstein-Bézier form (B-form) of the polynomial pieces, see Section 3.4. Given a tetrahedron $T := [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] \in \Delta$ with vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$, and $\mathbf{v}_3$, each $p \in \mathcal{P}_2$ has a unique representation

$$p_{|_T} \equiv \sum_{i+j+k+\ell=2} b_{ijk\ell} \, B_{ijk\ell}. \tag{5.1}$$

As usual, we associate the Bernstein-Bézier coefficients (B-coefficients) $b_{ijk\ell}$ of p in the form (5.1) with the domain points

$$\xi_{ijk\ell} := (i\mathbf{v}_0 + j\mathbf{v}_1 + k\mathbf{v}_2 + \ell\mathbf{v}_3)/2, \quad i+j+k+\ell = 2,$$

in T, and we call the points $(\xi, b_\xi) \in \mathbb{R}^4$ *control points* of p. Let $\mathcal{D}_\Delta := \mathcal{D}_2(\Delta) = \cup_{T\in\Delta}\mathcal{D}_T$ be the union of the sets of domain points associated with the tetrahedra of Δ. It is

easy to see that a continuous spline $s \in \mathcal{S}_2^0(\Delta)$ is uniquely defined by the coefficients $\{b_\xi : \xi \in \mathcal{D}_\Delta\}$ and the dimension of $\mathcal{S}_2^0(\Delta)$ is equal to the cardinality of $\mathcal{D}_\Delta$. Therefore, it follows that

$$\dim \mathcal{S}_2^0(\Delta) = \#\mathcal{D}_\Delta = V + E,$$

where V and E denote the vertices and edges of Δ. If $s \in C^1(\Omega)$, then these coefficients cannot be chosen arbitrarily, but have to satisfy certain smoothness conditions, see Section 3.4.4. Let $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ and $\widetilde{T} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \tilde{\mathbf{v}}_3]$ be two adjacent tetrahedra sharing a common triangular face $F := T \cap \widetilde{T} = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2]$ and let s be a quadratic continuous spline on $T \cup \widetilde{T}$ in its piecewise B-form, see Equation (5.1): $s|_T = p$ and $s|_{\widetilde{T}} = \tilde{p}$. Then s is C^1-smooth across F if and only if

$$\tilde{b}_{ijk1} = b_{i+1,j,k,0}\ \phi_0(\tilde{\mathbf{v}}_3) + b_{i,j+1,k,0}\ \phi_1(\tilde{\mathbf{v}}_3) + b_{i,j,k+1,0}\ \phi_2(\tilde{\mathbf{v}}_3) + b_{i,j,k,1}\ \phi_3(\tilde{\mathbf{v}}_3), \qquad (5.2)$$

where $i + j + k = 1$. These trivariate smoothness conditions can degenerate to lower dimensional conditions if one or two of the barycentric coordinates vanish at the point $\tilde{\mathbf{v}}_3$, see Section 3.4.4. The appearance of these degenerated conditions is typical for splines on uniform partitions (see [HNR*04, HNZ06]). Especially spline spaces with low polynomial degree are highly complex, due to the fact that the smoothness conditions (5.2) have to be simultaneously satisfied for all interior faces of Δ. We conclude this section with two further definitions. Let $\mathbf{v} \in V$ be a vertex of Δ. We define by $D^1(\mathbf{v})$ the *ball* with radius 1 around $\mathbf{v}$. Note that $D^1(\mathbf{v})$ corresponds to the set of midpoints emanating from $\mathbf{v}$ and $\mathbf{v}$ itself. An easy geometric interpretation of Equation (5.2) shows that a spline $s \in \mathcal{S}_2^0(\Delta)$ is C^1-smooth in a vertex $\mathbf{v}$ iff all control points (ξ, b_ξ), $\xi \in D^1(\mathbf{v})$ lie in the same hyper plane in $\mathbb{R}^4$. Let $\mathcal{M} \subseteq \mathcal{D}_\Delta$ and b_ξ, $\xi \in \mathcal{M}$, be the associated B-coefficients of a spline $s \in \mathcal{S}_2^1(\Delta)$. Then, we call $\mathcal{M}$ a determining set for the space $\mathcal{S}_2^1(\Delta)$ if $b_\xi = 0$, $\xi \in \mathcal{M}$, implies $s \equiv 0$. In the next section we will use these results to analyze quadratic C^1-splines on a special type of uniform tetrahedral partitions, namely quadratic C^1-splines on truncated octahedral partitions.

5.2 Smoothness of Quadratic Splines on Truncated Octahedral Partitions

Let $\mathcal{VD}$ be the Voronoi diagram of the Body Centered Cubic lattice (BCC lattice). The BCC lattice is obtained from a regular cubic grid by inserting additional grid points at the center of each of the grid cells. The Voronoi cells of $\mathcal{VD}$ are truncated octahedra and we note that it is possible to tessellate the $\mathbb{R}^3$ by a uniform partition of truncated octahedra centered at the vertices of the BCC lattice. A truncated octahedron is an Archimedean solid with 14 faces consisting of 6 squares and 8 regular hexagons, see Figure 5.1. The 24 vertices of a truncated octahedron $\mathcal{T}$ with radius $h \in \mathbb{N}$ centered at the origin can be described by all permutations of $(0, \pm h/2, \pm h)^\mathsf{T}$.

Let $\Diamond$ be a truncated octahedral partition and $\Omega \subset \mathbb{R}^3$ the covered region in $\mathbb{R}^3$. We get a tetrahedral partition Δ of Ω by subdividing all $\mathcal{T} \in \Diamond$ in the following way. We insert a vertex at the center of all faces and edges of $\mathcal{T}$, as well as at the center of $\mathcal{T}$

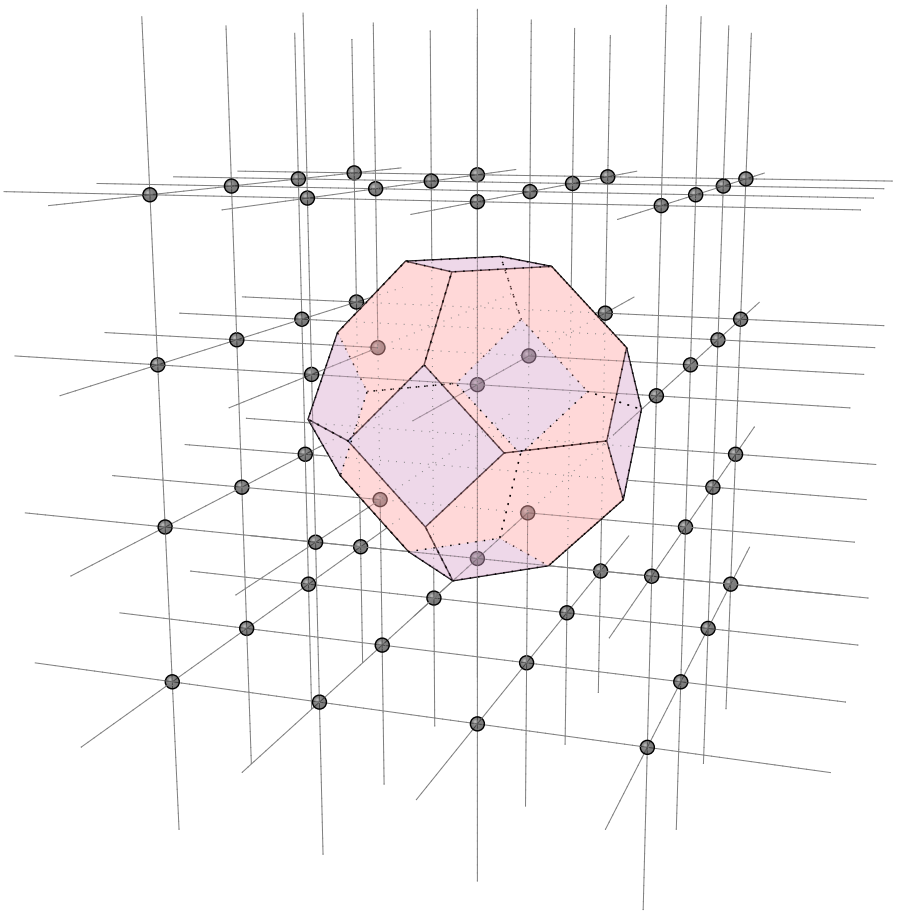

Figure 5.1: A truncated octahedron $\mathcal{T}$ embedded in the cubic grid. The data points (black dots) are located on the center of each hexagonal face.

itself. Then we triangulate the faces of $\mathcal{T}$ by connecting all vertices on the boundary of the faces with the vertex in the center of the face. Finally we connect all vertices on the boundary of $\mathcal{T}$ with the vertex in the center of $\mathcal{T}$ and achieve a tetrahedral partition Δ, where every truncated octahedron $\mathcal{T} \in \Diamond$ is split into 144 tetrahedra. Each tetrahedron $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] \in \Delta$ has one vertex, say $\mathbf{v}_0$ at the center of a truncated octahedron $\mathcal{T} \in \Diamond$, one vertex, say $\mathbf{v}_1$ at the center of a face of $\mathcal{T}$, another vertex, say $\mathbf{v}_2$ at the midpoint of one of the edges of $\mathcal{T}$ and the remaining vertex $\mathbf{v}_3$ is a vertex of $\mathcal{T}$. Thereby, the triangular face $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ lies on a face of $\mathcal{T}$. Since the faces of $\mathcal{T}$ are either squares or hexagons this split results in two differently shaped tetrahedra, according to the face they are associated with. We say $T \in T^S$ if $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ lies on a square face of $\mathcal{T}$ and $T \in T^H$ if $[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$ lies on a hexagonal face of $\mathcal{T}$. In the following we want to describe the C^1-smoothness conditions of a spline $s \in \mathcal{S}_2^1(\Delta)$ across the interior triangular faces of Δ. We use the notation $F_\nu = [\mathbf{v}_{\mu_1}, \mathbf{v}_{\mu_2}, \mathbf{v}_{\mu_3}]$, $\nu = 0, \ldots, 3$, $\mu_1, \mu_2, \mu_3 \in \{0, \ldots, 3\} \backslash \{\nu\}$, $\mu_1 < \mu_2 < \mu_3$. Here, F_ν, $\nu = 0, \ldots, 3$, is the triangular face of T opposing the vertex $\mathbf{v}_\nu$.

Further, we denote the triangular faces of a tetrahedron $T \in \Delta$ by F_ν^S, $\nu = 0, \dots, 3$, if $T \in T^S$ and by F_ν^H, $\nu = 0, \dots, 3$, otherwise. Note that the triangular face F_1^S is the intersection between two adjacent tetrahedra of T^S and T^H, while the triangular face F_1^H is the intersection between two adjacent tetrahedra of T^H only. Let T and $\widetilde{T}$ be two tetrahedra sharing a triangular face and $b_{ijk\ell}$ and $\tilde{b}_{ijk\ell}$ the corresponding coefficients of the piecewise polynomials $s|_T = p$ and $s|_{\widetilde{T}} = \tilde{p}$, respectively. In the case $T \cap \widetilde{T} = F_1^S$ we identify the tetrahedra by $T \in T^S$ and $\widetilde{T} \in T^H$, while in all other cases the conditions are completely symmetric. By using Equation (5.2) and some elementary computations, we obtain that $s \in \mathcal{S}_2^1(\Delta)$ iff the following conditions for its coefficients are satisfied:

- **Smoothness across F_0^S and F_0^H**

$$b_{1100} = 2b_{0200} - \tilde{b}_{1100}, \tag{5.3a}$$
$$b_{1010} = 2b_{0110} - \tilde{b}_{1010}, \tag{5.3b}$$
$$b_{1001} = 2b_{0101} - \tilde{b}_{1001}. \tag{5.3c}$$

- **Smoothness across F_3^S and F_3^H**

$$b_{1001} = 2b_{1010} - \tilde{b}_{1001}, \tag{5.4a}$$
$$b_{0101} = 2b_{0110} - \tilde{b}_{0101}, \tag{5.4b}$$
$$b_{0011} = 2b_{0020} - \tilde{b}_{0011}. \tag{5.4c}$$

- **Smoothness across F_2^S**

$$b_{1010} = b_{1100} - \tilde{b}_{1010} + b_{1001}, \tag{5.5a}$$
$$b_{0110} = b_{0200} - \tilde{b}_{0110} + b_{0101}, \tag{5.5b}$$
$$b_{0011} = b_{0101} - \tilde{b}_{0011} + b_{0002}. \tag{5.5c}$$

- **Smoothness across F_2^H**

$$b_{1010} = \frac{1}{2}b_{1100} - \tilde{b}_{1010} + \frac{3}{2}b_{1001}, \tag{5.6a}$$
$$b_{0110} = \frac{1}{2}b_{0200} - \tilde{b}_{0110} + \frac{3}{2}b_{0101}, \tag{5.6b}$$
$$b_{0011} = \frac{1}{2}b_{0101} - \tilde{b}_{0011} + \frac{3}{2}b_{0002}. \tag{5.6c}$$

- **Smoothness across F_1^S**

$$b_{1100} = \frac{1}{3}b_{2000} - \frac{2}{3}\tilde{b}_{1100} + \frac{4}{3}b_{1010}, \tag{5.7a}$$
$$b_{0110} = \frac{1}{3}b_{1010} - \frac{2}{3}\tilde{b}_{0110} + \frac{4}{3}b_{0020}, \tag{5.7b}$$
$$b_{0101} = \frac{1}{3}b_{1001} - \frac{2}{3}\tilde{b}_{0101} + \frac{4}{3}b_{0011}. \tag{5.7c}$$

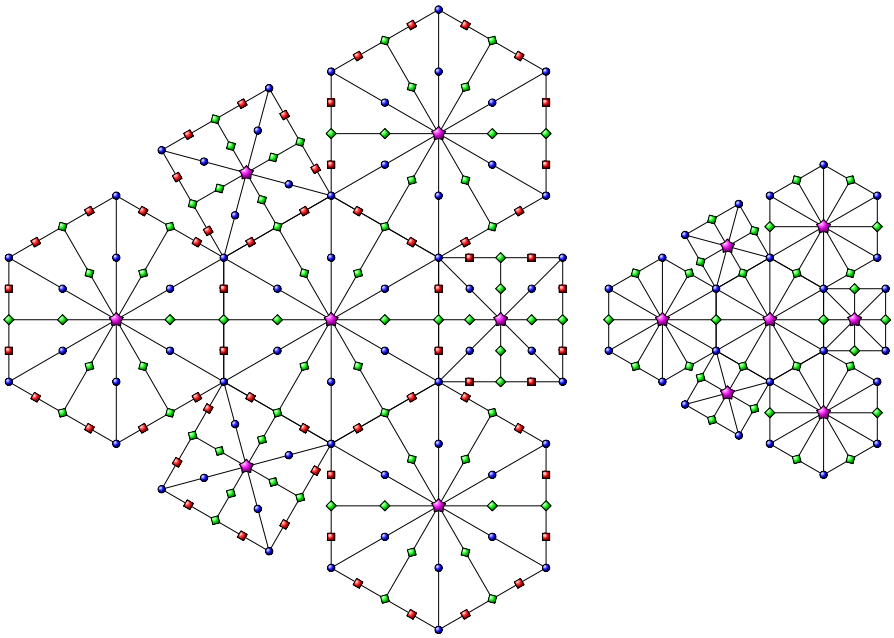

Figure 5.2: *Left*: coefficients on the outer layer of the half of an unfolded truncated octahedron $\mathcal{T}$. *Right*: inner layer of $\mathcal{T}$. 'Red' squares denote the determining set $\mathcal{M}$ on $\mathcal{T}$. The other coefficients involved in the smoothness conditions (5.3-5.8) are shown as 'blue' dots, 'green' diamonds and 'magenta' stars.

• **Smoothness across F_1^H**

$$b_{1100} = \frac{2}{3}b_{2000} - \tilde{b}_{1100} + \frac{4}{3}b_{1010}, \tag{5.8a}$$

$$b_{0110} = \frac{2}{3}b_{1010} - \tilde{b}_{0110} + \frac{4}{3}b_{0020}, \tag{5.8b}$$

$$b_{0101} = \frac{2}{3}b_{1001} - \tilde{b}_{0101} + \frac{4}{3}b_{0011}. \tag{5.8c}$$

Note that Equation (5.3) describes the smoothness conditions across the triangular faces between neighboring truncated octahedra and (5.4-5.8) characterize the smoothness across triangular faces inside a single truncated octahedron. Since the smoothness conditions across the faces F_0 and F_3 degenerate to univariate conditions, we can describe these conditions simultaneously for the cases F_0^S and F_0^H, and F_3^S and F_3^H, respectively. The following Lemma gives a determining set for the space $\mathcal{S}_2^1(\Delta)$. Here, T_B denotes tetrahedra in Δ with the vertex $\mathbf{v}_3$ lying on the boundary of Ω.

Lemma 1. *The set* $\mathcal{M} := \{\xi_{0011}^T, \xi_{0002}^{T_B} : T, T_B \in \Delta\}$ *is a determining set of the space* $\mathcal{S}_2^1(\Delta)$.

Proof. We have to show that for any spline $s \in \mathcal{S}_2^1(\Delta)$, with $b_\xi = 0$, $\xi \in \mathcal{M}$, it follows that $s \equiv 0$. Let $T \in \Delta$, $\mathbf{v}_3 \in T$ and $D^1(\mathbf{v}_3)$ be the ball with radius 1 around $\mathbf{v}_3$. The spline $s \in \mathcal{S}_2^0(\Delta)$ is C^1-continuous in $\mathbf{v}_3$ iff all control points (ξ, b_ξ), $\xi \in D^1(\mathbf{v}_3)$, lie

in the same hyper plane in $\mathbb{R}^4$. Since $\mathcal{M} \cap D^1(\mathbf{v}_3)$ contains exactly four points which do not lie on a plane in $\mathbb{R}^3$, the four control points (ξ, b_ξ), $\xi \in \mathcal{M} \cap D^1(\mathbf{v}_3)$ uniquely determine a hyper plane in $\mathbb{R}^4$ and it follows that $b_\xi = 0$, for all $\xi \in D^1(\mathbf{v}_3)$. This also follows by combining the smoothness conditions (5.3c), (5.5c), (5.6c), (5.7c) and (5.8c), and some elementary computations. Therefore, $b_\xi = 0$, for all $\xi = \xi^T_{i,j,k,\ell} \in \mathcal{D}_\Delta$, $\ell \geq 1$. Now the smoothness conditions (5.4a-c) imply that $b_\xi = 0$, for all $\xi = \xi^T_{i,j,k,\ell} \in \mathcal{D}_\Delta$, $k \geq 1$. Furthermore, the conditions (5.5a-b) and (5.6a-b) then imply $b_\xi = 0$, for all $\xi = \xi^T_{i,j,k,\ell} \in \mathcal{D}_\Delta$, $j \geq 1$. Finally, conditions (5.7a) and (5.8a) imply $b_\xi = 0$, for all $\xi = \xi^T_{i,j,k,\ell} \in \mathcal{D}_\Delta$, $(i,j,k,\ell) = (2,0,0,0)$. Therefore it follows that $b_\xi = 0$, for all $\xi \in \mathcal{D}_\Delta$ and $s \equiv 0$. □

Figure 5.2 shows the order in which the coefficients in the proof of Lemma 1 are determined. Due to symmetry, only one half of an unfolded truncated octahedron is shown. On the left side are the points on the boundary of the truncated octahedron and on the right side the points at distance one to the center vertex $\mathbf{v}_0$. We assume, that the truncated octahedron does not have any boundary vertices $\mathbf{v}_3 \in T_B$. The points of the determining set $\mathcal{M}$ are shown as 'red' squares (■), while the points associated with the coefficients directly determined through the hyper plane conditions and the coefficients b_ξ, $\xi \in \mathcal{M}$, are shown as 'blue' dots (●). Finally, 'green' diamonds (◆) and 'magenta' stars (★) show the points which associated coefficients are determined through the univariate conditions (5.4a-c) and bivariate conditions (5.5a-b) and (5.6a-b), respectively.

5.3 The Quasi-Interpolating Scheme

In this section we describe the construction of a family of quasi-interpolation operators on truncated octahedral partitions. We give a detailed description of the quasi-interpolation scheme and prove our main result, as well as some useful properties of certain operators. For $n \in \mathbb{N}$, $n \geq 3$, let

$$\mathcal{V} := \{\mathbf{v}_{ijk} = (ih, jh, kh)^\mathsf{T} : i,j,k = -1, \ldots, (n+1)\}$$

be the cubic grid of $(n+3)^3$ points with grid size $h \in \mathbb{R}$ and $\Diamond$ the truncated octahedral partition of truncated octahedra with radius h centered at the vertices of the BCC grid

$$\widetilde{\mathcal{V}} := \{\mathbf{v}_{ijk} + (\tfrac{h}{2}, \tfrac{h}{2}, \tfrac{h}{2})^\mathsf{T} : i,j,k = 1, \ldots, (n-2),\ i,j,k \text{ all odd or } i,j,k \text{ all even}\}.$$

We assume that the values of a function $f \in C(\Omega^*)$, $\Omega^* = [-h, (n+1)h]^3 \subseteq \mathbb{R}^3$ are known at the grid points of $\mathcal{V}$. In each truncated octahedron $\mathcal{T} \in \Diamond$ lie exactly 8 points $\mathbf{z}^\mathcal{T}_1, \ldots, \mathbf{z}^\mathcal{T}_8$ of the set $\mathcal{V}$ located on the hexagonal faces of $\mathcal{T}$, see Figure 5.1, which we denote by $\mathcal{V}_\mathcal{T} := \{\mathbf{z}^\mathcal{T}_1, \ldots, \mathbf{z}^\mathcal{T}_8\}$. Although, $\mathcal{V}$ contains data points that do not lie in $\Diamond$, we assume in the following that these data points are also associated with truncated octahedra. Let Δ be the tetrahedral partition of $\Diamond$ described in the previous section and $\Omega \subset \Omega^*$ the covered region in $\mathbb{R}^3$. In the following we describe our method to determine an approximating quadratic spline s_f on Δ by setting all B-coefficients b_ξ, $\xi \in \mathcal{D}_\Delta$, to

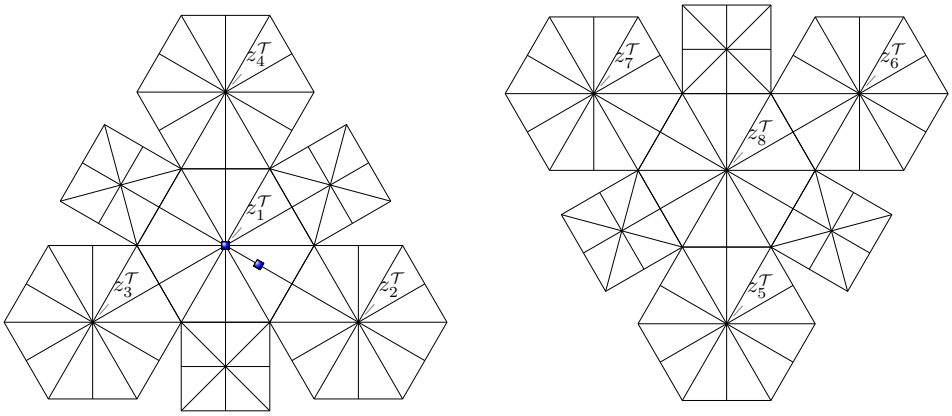

Figure 5.3: The typical order of the data points $\mathbf{z}_1^{\mathcal{T}}, \ldots, \mathbf{z}_8^{\mathcal{T}}$ for the coefficients b_ξ, $\xi = \xi_{0200}$ and $\xi = \xi_{0110}$ ('blue' squares) on the outer layer of an unfolded truncated octahedron $\mathcal{T}$.

appropriate averages of the data. Thereby, for the calculation of each coefficient b_ξ we use only local data portions in the following sense.

If the domain point ξ lies in the interior of a truncated octahedron $\mathcal{A} \in \Diamond$ then the associated coefficient b_ξ is determined by weighted averages of the 8 data values $A_i := f(\mathbf{z}_i^{\mathcal{A}})$, $\mathbf{z}_i^{\mathcal{A}} \in \mathcal{V}_{\mathcal{A}}$:

$$b_\xi = \sum_{i=1,\ldots,8} w_i^A A_i, \quad w_i^A \in \mathbb{R}.$$

Here, as well as in the following, the data points $\mathbf{z}_i^{\mathcal{T}}$, $i = 1, \ldots, 8$, of a truncated octahedron $\mathcal{T}$ are ordered by increasing (Euclidean) distances to the point ξ. This means $\mathbf{z}_1^{\mathcal{T}} = \min_{\mathbf{z} \in \mathcal{V}_{\mathcal{T}}} ||\xi - \mathbf{z}||_2$ and $\mathbf{z}_8^{\mathcal{T}} = \max_{\mathbf{z} \in \mathcal{V}_{\mathcal{T}}} ||\xi - \mathbf{z}||_2$, respectively. In the case of equal distance the order of the points can be chosen arbitrarily. Figures 5.3 and 5.4 show a typical order of the data points for a selection of domain points on the outer layer of an unfolded truncated octahedron $\mathcal{T}$.

If the domain point ξ lies in the interior of a face of a truncated octahedron then the associated coefficient b_ξ is determined by weighted averages of the data values regarding the two truncated octahedra sharing this face. Let $\mathcal{A}, \mathcal{B} \in \Diamond$ be two truncated octahedra sharing a face, then b_ξ, $\xi \in \mathrm{int}(\mathcal{A} \cap \mathcal{B})$, is determined by

$$b_\xi = \sum_{i=1,\ldots,8} w_i^A A_i + \sum_{i=1,\ldots,8} w_i^B B_i, \quad w_i^A, w_i^B \in \mathbb{R},$$

where $B_i := f(\mathbf{z}_i^{\mathcal{B}})$, $\mathbf{z}_i^{\mathcal{B}} \in \mathcal{V}_{\mathcal{B}}$. Here, $\mathcal{A}$ denotes the truncated octahedron with $||\xi - \mathbf{z}_2^{\mathcal{A}}||_2 \leq ||\xi - \mathbf{z}_2^{\mathcal{B}}||_2$. If $\mathcal{A}$ and $\mathcal{B}$ share a square face we have equal distances and the notation can be chosen arbitrarily. In the case that $\mathcal{A}$ and $\mathcal{B}$ share a hexagonal face, they have the common data point $\mathbf{z}_1^{\mathcal{A}} = \mathbf{z}_1^{\mathcal{B}}$ and therefore the data value $A_1 = B_1$ is weighted by $w_1^A + w_1^B$.

If the domain point ξ lies in the interior of an edge of a truncated octahedron then the associated coefficient b_ξ is determined by weighted averages of the data values regarding

the three truncated octahedra sharing this edge. Let $\mathcal{A}, \mathcal{B}, \mathcal{C} \in \Diamond$ be three truncated octahedra sharing an edge, then b_ξ, $\xi \in \text{int}(\mathcal{A} \cap \mathcal{B} \cap \mathcal{C})$, is determined by

$$b_\xi = \sum_{i=1,\ldots,8} w_i^A A_i + \sum_{i=1,\ldots,8} w_i^B B_i + \sum_{i=1,\ldots,8} w_i^C C_i,$$

where $w_i^A, w_i^B, w_i^C \in \mathbb{R}$, and $C_i := f(\mathbf{z}_i^{\mathcal{C}})$, $\mathbf{z}_i^{\mathcal{C}} \in \mathcal{V}_{\mathcal{C}}$. Here, $\mathcal{A}$ denotes again the truncated octahedron with $||\xi - \mathbf{z}_2^{\mathcal{A}}||_2 = \min_{\mathcal{T} \in \Diamond} ||\xi - \mathbf{z}_2^{\mathcal{T}}||_2$. Since $\mathcal{A}$ and $\mathcal{B}$, as well as $\mathcal{A}$ and $\mathcal{C}$, share a hexagonal face, they have the common data points $\mathbf{z}_1^{\mathcal{A}} = \mathbf{z}_1^{\mathcal{B}}$ and $\mathbf{z}_2^{\mathcal{A}} = \mathbf{z}_1^{\mathcal{C}}$. The data values $A^1 = B^1$ and $A^2 = C^1$ are then weighted by $w_1^A + w_1^B$ and $w_2^A + w_1^C$, respectively.

Finally, if the domain point ξ lies on a vertex of a truncated octahedron then the associated coefficient b_ξ is determined by weighted averages of the data values regarding the four truncated octahedra sharing this vertex. Let $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D} \in \Diamond$ be the four truncated octahedra sharing the vertex v, then b_ξ, $\xi = v$, is determined by

$$b_\xi = \sum_{i=1,\ldots,8} w_i^A A_i + \sum_{i=1,\ldots,8} w_i^B B_i + \sum_{i=1,\ldots,8} w_i^C C_i + \sum_{i=1,\ldots,8} w_i^D D_i,$$

where $w_i^A, w_i^B, w_i^C, w_i^D \in \mathbb{R}$, and $D_i := f(\mathbf{z}_i^{\mathcal{D}})$, $\mathbf{z}_i^{\mathcal{D}} \in \mathcal{V}_{\mathcal{D}}$. Due to symmetry the notation can be chosen arbitrarily here. Since exactly four hexagonal faces share this vertex, we also have common data points in this case. Without loss of generality we say $\mathbf{z}_1^{\mathcal{A}} = \mathbf{z}_1^{\mathcal{B}}$, $\mathbf{z}_2^{\mathcal{A}} = \mathbf{z}_1^{\mathcal{C}}$, $\mathbf{z}_1^{\mathcal{D}} = \mathbf{z}_2^{\mathcal{B}}$ and $\mathbf{z}_2^{\mathcal{D}} = \mathbf{z}_2^{\mathcal{C}}$.

In the following we describe a family of quasi-interpolation operators $\mathcal{Q}_k : C(\Omega^*) \to S_2^0(\Delta), k \geq 1$ by giving the calculation rules for the B-coefficients of $s_f := \mathcal{Q}_k(f)$ in its piecewise representation (5.1) explicitly. Here, we assume that the data values come from a continuous function $f \in C(\Omega^*)$. In some cases we have to distinguish which type of tetrahedra the coefficients are associated with. We denote by b_ξ^S B-coefficients associated with domain-points $\xi \in T$, where T is in T^S and by b_ξ^H all other B-coefficients. Since often the same weights appear we use the compact notation $AB_{1,2} := A_1 + B_1 + A_2 + B_2$ (analogous for $A_{1,2}, ABCD_{1,2}$, etc.). The determination of the B-coefficients is as follows:

$$\begin{aligned}
b_{0011} := \frac{1}{k} 2^{-6} \; & (\; 2 \; (\; (k+3)A_{1,2} + (k+1)A_{3,4} + (k-1)A_{5,6} + (k-3)A_{7,8} \;) \\
& \quad + 3 \; (\; (k+3)BC_{1,2} + (k+1)BC_{3,4} \\
& \qquad + (k-1)BC_{5,6} + (k-3)BC_{7,8} \;) \;) \\
b_{0020} := \frac{1}{k} 2^{-6} \; & (\; 2 \; (\; (k+3)A_{1,2} + kA_{3,4,5,6} + (k-3)A_{7,8} \;) \\
& \quad + 3 \; (\; (k+3)BC_1 + (k+2)BC_{2,3} + (k+1)BC_4 \\
& \qquad + (k-1)BC_5 + (k-2)BC_{6,7} + (k-3)BC_8 \;) \;) \\
b_{0002} := \frac{1}{k} 2^{-5} \; & (\; (k+3)ABCD_{1,2} + (k+1)ABCD_{3,4} \\
& \quad + (k-1)ABCD_{5,6} + (k-3)ABCD_{7,8} \;) \\
b_{0101}^{S,H} := \frac{1}{k} 2^{-4} \; & (\; (k+3)AB_{1,2} + (k+1)AB_{3,4} + (k-1)AB_{5,6} + (k-3)AB_{7,8} \;)
\end{aligned}$$

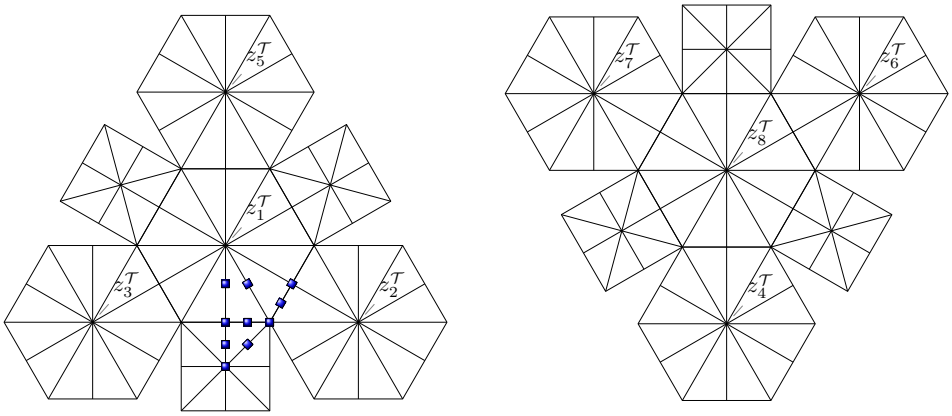

Figure 5.4: The typical order of the data points $\mathbf{z}_1^{\mathcal{T}}, \ldots, \mathbf{z}_8^{\mathcal{T}}$ for the remaining coefficients b_ξ ('blue' squares) on the outer layer of an unfolded truncated octahedron $\mathcal{T}$.

$$
\begin{aligned}
b^S_{0110} &:= \frac{1}{k}2^{-4}\ (\ (k+3)AB_1 + (k+2)AB_{2,3} + (k+1)AB_4 \\
&\qquad + (k-1)AB_5 + (k-2)AB_{6,7} + (k-3)AB_8\ ) \\
b^H_{0110} &:= \frac{1}{k}2^{-4}\ (\ (k+3)A_{1,2} + kA_{3,4,5,6} + (k-3)A_{7,8} \\
&\qquad + (k+3)B_1 + (k+2)B_{2,3} + (k+1)B_4 \\
&\qquad + (k-1)B_5 + (k-2)B_{6,7} + (k-3)B_8\ ) \\
b^S_{0200} &:= \frac{1}{k}2^{-4}\ (\ (k+2)AB_{1,2,3,4} + (k-2)AB_{5,6,7,8}\ ) \\
b^H_{0200} &:= \frac{1}{k}2^{-4}\ (\ (k+3)AB_1 + (k+1)AB_{2,3,4} + (k-1)AB_{5,6,7} + (k-3)AB_8\ ) \\
b_{1001} &:= \frac{1}{k}2^{-3}\ (\ (k+3)A_{1,2} + (k+1)A_{3,4} + (k-1)A_{5,6} + (k-3)A_{7,8}\ ) \\
b^S_{1010} &:= \frac{1}{k}2^{-3}\ (\ (k+3)A_1 + (k+2)A_{2,3} + (k+1)A_4 \\
&\qquad + (k-1)A_5 + (k-2)A_{6,7} + (k-3)A_8\ ) \\
b^H_{1010} &:= \frac{1}{k}2^{-3}\ (\ (k+3)A_{1,2} + kA_{3,4,5,6} + (k-3)A_{7,8}\ ) \\
b^S_{1100} &:= \frac{1}{k}2^{-3}\ (\ (k+2)A_{1,2,3,4} + (k-2)A_{5,6,7,8}\ ) \\
b^H_{1100} &:= \frac{1}{k}2^{-3}\ (\ (k+3)A_1 + (k+1)A_{2,3,4} + (k-1)A_{5,6,7} + (k-3)A_8\ ) \\
b_{2000} &:= \ 2^{-3}\ A_{1,2,3,4,5,6,7,8}
\end{aligned}
\tag{5.9}
$$

We note that all coefficients b_ξ, $\xi \in \mathcal{D}_\Delta$, are uniquely determined and therefore the spline s_f is in $\mathcal{S}_2^0(\Delta)$. The next result will show that the weights are chosen carefully, so that s_f is indeed globally C^1 on Ω.

Theorem 2. *The quasi-interpolation operator $\mathcal{Q}_k : C(\Omega^*) \to \mathcal{S}_2^0(\Delta)$, $k \geq 1$ is a linear*

mapping into the space $\mathcal{S}_2^1(\Delta)$.

Proof. We have to show that all conditions (5.3-5.8) are simultaneously satisfied. Since the complete proof is large and involves only elementary computations, we show the procedure exemplary on two typical conditions. At first we consider condition (5.3a) of univariate type in the case that F_0^S lies on a common square face of two adjacent truncated octahedra:

$$b_{1100}^S + \tilde{b}_{1100}^S = 2b_{0200}^S.$$

Now we apply Equation (5.9) for the involved coefficients and obtain on the left-hand side of the equation

$$\frac{1}{k}2^{-3}\ (\ (k+2)A\widetilde{A}_{1,2,3,4} + (k-2)A\widetilde{A}_{5,6,7,8}\ ).$$

Since $\widetilde{\mathcal{A}} = \mathcal{B}$ this is exactly two times the coefficient b_{0200}^S and the condition is satisfied. Next, we consider condition (5.6a) of bivariate type in the case that F_2^H is the common triangular face of two adjacent tetrahedra $T \in T^H$ and $\widetilde{T} \in T^H$ inside a truncated octahedron:

$$b_{1010}^S + \tilde{b}_{1010}^H = \frac{1}{2}b_{1100}^H + \frac{3}{2}b_{1001}.$$

We note that one of the coefficients on the left-hand side is of type b_ξ^S and the other one of type b_ξ^H. Without loss of generality we say $b_{1010} = b_{1010}^S$ and $\tilde{b}_{1010} = \tilde{b}_{1010}^H$. Now we apply condition (5.9) for the involved coefficients on the left-hand side of the equation and considering that $\widetilde{\mathcal{A}} = \mathcal{A}$ we obtain

$$\begin{aligned}\frac{1}{k}2^{-3}\ (\ &(2k+6)A_1 + (2k+5)A_2 + (2k+2)A_3 + (2k+1)A_4 \\ &+ (2k-1)A_5 + (2k-2)A_6 + (2k-5)A_7 + (2k-6)A_8\ ).\end{aligned}$$

If we apply (5.9) to the coefficients on the right-hand side of the equation, we have to pay attention that only b_{1001} has the same ordering of the data points as the coefficients b_{1010}^S and $\tilde{b}_{1010}^H$. Therefore, for now we denote the data values regarding b_{1100}^H by $\hat{A}_1, \ldots, \hat{A}_8$ and obtain

$$\begin{aligned}\frac{1}{k}2^{-4}\ (\ &(k+3)\hat{A}_1 + (k+1)\hat{A}_{2,3,4} + (k-1)\hat{A}_{5,6,7} + (k-3)\hat{A}_8 \\ &+ 3(\ (k+3)A_{1,2} + (k+1)A_{3,4} + (k-1)A_{5,6} + (k-3)A_{7,8}\ )\ ).\end{aligned}$$

Considering the order of the data points we achieve the following correlations between the data values:

$$\hat{A}_1 = A_1,\ \hat{A}_2 = A_2,\ \hat{A}_3 = A_3,\ \hat{A}_4 = A_5,\ \hat{A}_5 = A_4,\ \hat{A}_6 = A_6,\ \hat{A}_7 = A_7,\ \hat{A}_8 = A_8.$$

Applying this to the above term, we obtain

$$\begin{aligned}\frac{1}{k}2^{-4}\ (\ &(4k+12)A_1 + (4k+10)A_2 + (4k+4)A_3 + (4k+2)A_4 \\ &+ (4k-2)A_5 + (4k-4)A_6 + (4k-10)A_7 + (4k-12)A_8\ )\end{aligned}$$

and condition (5.6a) is satisfied. All other conditions (5.3-5.8) can be shown similarly. Only the order of the data points has to be taken into account for each coefficient individually. □

We conclude this section with some important properties of the quasi-interpolation operators.

Corollary 3. *The quasi-interpolation operator* $\mathcal{Q}_k : C(\Omega^*) \to \mathcal{S}_2^1(\Delta)$ *is positive for* $k \geq 3$.

Proof. For $k \geq 3$ only positive weights of the data values appear in the calculation of the B-coefficients in (5.9). Therefore, assuming that the data values come from a positive function $f \in C(\Omega^*)$, the quasi-interpolating spline $s_f := \mathcal{Q}_k(f)$ is also positive. This follows directly from the piecewise polynomial representation (5.1) and the non-negativity of the Bernstein polynomials. Finally, it is easy to see that $f \equiv 0$ implies $s_f \equiv 0$. □

The next Lemma shows that the quasi-interpolation operators $\mathcal{Q}_k$, $k \geq 1$, reproduce certain polynomials which is essential for their approximation properties, see Section 5.4. In particular, the operator $\mathcal{Q}_2$ reproduces quadratic polynomials up to a constant.

Lemma 4. *The following statements hold.*

$$
\begin{aligned}
&(i) && \mathcal{Q}_k(1) \equiv 1, \ \forall k \geq 1 \\
&(ii) && \mathcal{Q}_2(p) \equiv p, \ \forall p \in \mathcal{P}_1 \\
&(iii) && \mathcal{Q}_2(p) \equiv p, \ p \in \{xy, xz, yz\} \\
&(iv) && \mathcal{Q}_2(p) \equiv p + \tfrac{h^2}{4}, \ p \in \{x^2, y^2, z^2\} \\
&(v) && \text{if } p \in \mathcal{P}_2, \text{ then there exists a constant } C \text{ independent of } h, \\
& && \text{such that } \mathcal{Q}_2(p) \equiv p + Ch^2
\end{aligned}
$$

Proof. (i) If all data values in the Equation (5.9) are equal to 1 it is easy to see that also all coefficients b_ξ are equal to 1. Therefore, it follows directly from the partition of unity of the B-polynomials, see Equation (3.10), that $\mathcal{Q}_k(1) \equiv 1$, for all $k \geq 1$.
For (ii), we have to show that the quasi-interpolation operator $\mathcal{Q}_2$ additionally to (i) reproduces the monomials $\{x, y, z\}$. Since Lemma 1 gives a determining set $\mathcal{M}$ for the space $\mathcal{S}_2^1(\Delta)$ it is sufficient to prove the reproduction for the coefficients b_ξ, $\xi \in \mathcal{M}$, only. Moreover, with (i) and the linearity of $\mathcal{Q}_2$, the reproduction is independent from the choice of the origin. Therefore we consider the tetrahedron $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] \in \Delta$, where $\mathbf{v}_0 = (0,0,0)^\mathsf{T}$, $\mathbf{v}_1 = (h,0,0)^\mathsf{T}$, $\mathbf{v}_2 = (h, h/4, h/4)^\mathsf{T}$ and $\mathbf{v}_3 = (h, h/2, 0)^\mathsf{T}$. Here, T lies in a truncated octahedron with radius h centered at the origin. The coefficient b_ξ, $\xi = \xi_{0002}$, is only in $\mathcal{M}$ if ξ lies on the boundary of Δ and therefore we only need to consider the coefficient $b_\xi \in \mathcal{M}$, where $\xi = \xi_{0011}$. Let $p_1(x,y,z) := x$, $p_2(x,y,z) := y$ and $p_3(x,y,z) := z$. Then these functions are univariate polynomials restricted to the edge $[\mathbf{v}_2, \mathbf{v}_3]$ and the coefficients $b_{0011}^{\{i\}}$, $i = 1,2,3$, in their representation (5.1) regarding T can be computed by

$$
b_{0011}^{\{i\}} = 2p_i(\xi_{0011}) - \frac{1}{2}(p_i(\mathbf{v}_2) + p_i(\mathbf{v}_3)), \ i = 1,2,3.
$$

We obtain the following values for the coefficients $b_{0011}^{\{i\}}$, $i = 1, 2, 3$:

$$b_{0011}^{\{1\}} = h, \qquad b_{0011}^{\{2\}} = \frac{3}{8}h \qquad \text{and} \qquad b_{0011}^{[\{3\}]} = \frac{1}{8}h.$$

Now, we apply the order of increasing distances to the data points regarding the three truncated octahedra $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ with the common edge $[\mathbf{v}_2, \mathbf{v}_3]$ and obtain

$$\begin{array}{llll}
\mathbf{z}_1^A = (\frac{h}{2}, \frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_2^A = (\frac{3h}{2}, \frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_3^A = (\frac{h}{2}, \frac{3h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_4^A = (\frac{3h}{2}, \frac{3h}{2}, \frac{h}{2})^\mathsf{T}, \\
\mathbf{z}_5^A = (\frac{h}{2}, \frac{h}{2}, \frac{3h}{2})^\mathsf{T}, & \mathbf{z}_6^A = (\frac{3h}{2}, \frac{h}{2}, \frac{3h}{2})^\mathsf{T}, & \mathbf{z}_7^A = (\frac{h}{2}, \frac{3h}{2}, \frac{3h}{2})^\mathsf{T}, & \mathbf{z}_8^A = (\frac{3h}{2}, \frac{3h}{2}, \frac{3h}{2})^\mathsf{T}, \\
\mathbf{z}_1^B = (\frac{h}{2}, \frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_2^B = (\frac{h}{2}, \frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}, & \mathbf{z}_3^B = (\frac{h}{2}, \text{-}\frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_4^B = (\frac{h}{2}, \text{-}\frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}, \\
\mathbf{z}_5^B = (\text{-}\frac{h}{2}, \frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_6^B = (\text{-}\frac{h}{2}, \frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}, & \mathbf{z}_7^B = (\text{-}\frac{h}{2}, \text{-}\frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_8^B = (\text{-}\frac{h}{2}, \text{-}\frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}, \\
\mathbf{z}_1^C = (\frac{3h}{2}, \frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_2^C = (\frac{3h}{2}, \frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}, & \mathbf{z}_3^C = (\frac{3h}{2}, \text{-}\frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_4^C = (\frac{3h}{2}, \text{-}\frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}, \\
\mathbf{z}_5^C = (\frac{5h}{2}, \frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_6^C = (\frac{5h}{2}, \frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}, & \mathbf{z}_7^C = (\frac{5h}{2}, \text{-}\frac{h}{2}, \frac{h}{2})^\mathsf{T}, & \mathbf{z}_8^C = (\frac{5h}{2}, \text{-}\frac{h}{2}, \text{-}\frac{h}{2})^\mathsf{T}.
\end{array}$$

Here, $\mathcal{A}$ is the truncated octahedron with radius h centered at $(h, h, h)^\mathsf{T}$, while $\mathcal{B}$ and $\mathcal{C}$, are the truncated octahedra with radius h centered at $(0, 0, 0)^\mathsf{T}$ and $(2h, 0, 0)^\mathsf{T}$, respectively. Some elementary computations show that the calculation rule in Equation (5.9) for the coefficient b_{0011} indeed reproduces the coefficient $b_{0011}^{\{i\}}$ of p_i, for $i = 1, 2, 3$.
To prove (iii) and (iv) the procedure is similar. With (i), (ii) and the linearity of $\mathcal{Q}_2$, the reproduction is again independent from the choice of the origin. Now, a comparison of the calculated coefficient b_{0011} in (5.9) for $\mathcal{Q}_2(p)$, $p \in \{xy, xz, yz, x^2, y^2, z^2\}$ and the coefficient $b_{0011}^{\{p\}}$ of the polynomial p in its representation (5.1) regarding T shows that $b_{0011} = b_{0011}^{\{p\}}$ for $p \in \{xy, xz, yz\}$ and $b_{0011} = b_{0011}^{\{p\}} + \frac{h^2}{4}$ for $p \in \{x^2, y^2, z^2\}$. This proves (iii) and (iv), since $\mathcal{Q}_2$ is linear and reproduces constants.
Finally (v) follows directly by (i)-(iv) and the proof is complete. □

The next corollary shows that the derivatives of certain polynomials are also reproduced by the quasi-interpolation operator $\mathcal{Q}_2$. In particular, the exact reproduction of the derivatives of quadratic polynomials is the reason that this operator possesses the optimal approximation order for the derivatives of smooth functions. Here, we denote by $D_x^\alpha D_y^\beta D_z^\gamma(f)$, where $\alpha, \beta, \gamma \geq 0$, the higher order partial derivatives of a sufficiently smooth function f.

Corollary 5. *If $p \in \mathcal{P}_2$, then $D_x^\alpha D_y^\beta D_z^\gamma(\mathcal{Q}_2(p)) \equiv D_x^\alpha D_y^\beta D_z^\gamma(p)$, for $\alpha + \beta + \gamma \in \{1, 2\}$.*

Proof. Let $p \in \mathcal{P}_2$, then it follows from Lemma 4 (v), that

$$D_x^\alpha D_y^\beta D_z^\gamma(\mathcal{Q}_2(p)) \equiv D_x^\alpha D_y^\beta D_z^\gamma(p + Ch^2) \equiv D_x^\alpha D_y^\beta D_z^\gamma(p),$$

where $\alpha + \beta + \gamma \in \{1, 2\}$ and C is a constant independent of h. □

Although the quasi-interpolation operators $\mathcal{Q}_k$ do not reproduce linear polynomials for $k \neq 2$, it can be shown that all operators $\mathcal{Q}_k$, $k \geq 1$, reproduce the values of linear polynomials at the data points:

Lemma 6. *Let $p \in \mathcal{P}_1$ and $k \geq 1$, then $Q_k(p(\mathbf{v})) = p(\mathbf{v})$, all $\mathbf{v} \in \mathcal{V} \cap \Omega$.*

For a complete proof see [RK09]. The next result shows that the operators $\mathcal{Q}_k$, $k \geq 1$, are uniformly bounded. This is an important result, since it guarantees that the computation of the splines is a stable process. Here as well as in the following sections we use the ∞-norm defined by

$$\|f\|_\Gamma := \max_{u \in \Gamma} |f(u)|,$$

where Γ is a domain in $\mathbb{R}^3$ and f is a continuous function on Γ.

Lemma 7. *Let $f \in C(\Omega^*)$ and $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] \in \Delta$. Then we have*

$$||\mathcal{Q}_k(f)||_T \leq \frac{k+1}{k} ||f||_{\Omega_T}, \ k \geq 1,$$

where Ω_T is the cube with side length $\frac{3}{2}h$ centered in $\mathbf{v}_3$.

Proof. Since Ω_T contains all data points needed for the calculation of the B-coefficients b_ξ, $\xi \in \mathcal{D}_T$, in (5.9) and $k \geq 1$, we have

$$|b_\xi| \leq \max\{1, \frac{2k+1}{2k}, \frac{k+1}{k}\} ||f||_{\Omega_T}.$$

Then it follows from the non-negativity of the Bernstein polynomials and Equation (3.10) that

$$||\mathcal{Q}_k(f)||_T \leq \max\{|b_\xi| : \ \xi \in \mathcal{D}_T\}, \ k \geq 1$$

and the proof is complete. □

5.4 Approximation Properties

In this section, we analyze the approximation properties of the quasi-interpolating splines $\mathcal{Q}_k(f)$, $k \geq 1$. We use the same notations as in the previous section. Theorem 8 gives general error bounds for the approximation of the values of smooth functions for $k \geq 1$, and Theorem 9 gives special error bounds for the approximation of the values and derivatives of smooth functions for the case $k = 2$. In particular, we show that $\mathcal{Q}_2(f)$ approximates the values of a sufficiently smooth function with order two, while simultaneously the first and second derivatives are optimally approximated with order two and one, respectively. In the following we let

$$\|D^r(f)\|_\Gamma := \max\{\|D_x^\alpha D_y^\beta D_z^\gamma(f)\|_\Gamma : \alpha + \beta + \gamma = r\},$$

where Γ is a domain in $\mathbb{R}^3$ and $f \in C^r(\Gamma)$.

Theorem 8. *Let $f \in C(\Omega^*)$ and $\mathcal{Q}_k(f)$, $k \geq 1$, the quasi-interpolating spline defined in the previous section. Then the following statements hold.*

(i) If $f \in C^1(\Omega^*)$, then

$$||f - \mathcal{Q}_k(f)||_\Omega \leq C_k ||D^1(f)||_{\Omega^*} h,$$

where $C_k > 0$ is an absolute constant independent of f and h.

(ii) If $f \in C^2(\Omega^*)$, then

$$|(f - \mathcal{Q}_k(f))(v)| \leq \widetilde{C}_k ||D^2(f)||_{\Omega^*} h^2, \text{ all } \mathbf{v} \in \mathcal{V} \cap \Omega,$$

where $\widetilde{C}_k > 0$ is an absolute constant independent of f and h.

Proof. Let $f \in C^1(\Omega^*)$, $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] \in \Delta$ an arbitrary tetrahedron with vertices as in Section 5.2, Ω_T as in Lemma 7 and $p_f \in \mathcal{P}_0$ the constant Taylor polynomial of f at $\mathbf{v}_3$. Then there exists an absolute constant $C_T > 0$ independent of f and h such that

$$\|f - p_f\|_{\Omega_T} \leq C_T \|D^1(f)\|_{\Omega_T} h.$$

Furthermore, from the triangle inequality follows

$$\|f - \mathcal{Q}_k(f)\|_T \leq \|f - p_f\|_{\Omega_T} + \|\mathcal{Q}_k(f) - p_f\|_T.$$

With Lemma 4(i), the linearity of $\mathcal{Q}_k$ and Lemma 7 we obtain

$$\|\mathcal{Q}_k(f) - p_f\|_T = \|\mathcal{Q}_k(f - p_f)\|_T \leq \frac{k+1}{k} \|f - p_f\|_{\Omega_T}.$$

Combining these inequalities leads to

$$\|f - \mathcal{Q}_k(f)\|_T \leq C_k \|D^1(f)\|_{\Omega_T} h,$$

where $C_k = (1 + \frac{k+1}{k}) C_T$ is an absolute constant independent of f and h. This proves statement (i).
Now, let $f \in C^2(\Omega^*)$, $\mathbf{v} \in \mathcal{V} \cap \Omega$, $T = [\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3] \in \Delta$ a tetrahedron with $v_1 = v$, Ω_T as in Lemma 7 and $p_f \in \mathcal{P}_1$ the linear Taylor polynomial of f at $\mathbf{v}_3$. Then there exists an absolute constant $\widetilde{C}_T > 0$ independent of f and h such that

$$|(f - p_f)(\mathbf{v})| \leq \widetilde{C}_T \|D^2(f)\|_{\Omega_T} h^2.$$

Using the triangle inequality, we have

$$|(f - \mathcal{Q}_k(f))(\mathbf{v})| \leq |(f - p_f)(\mathbf{v})| + |(\mathcal{Q}_k(f) - p_f)(\mathbf{v})|$$

and with Lemma 6, the linearity of $\mathcal{Q}_k$ and Corollary 7 we obtain for the last term

$$|(\mathcal{Q}_k(f) - p_f)(\mathbf{v})| = |\mathcal{Q}_k(f - p_f)(\mathbf{v})| \leq \frac{k+1}{k} \|f - p_f\|_{\Omega_T}.$$

A combination of these inequalities leads to

$$|(f - \mathcal{Q}_k(f))(\mathbf{v})| \leq \widetilde{C}_k \|D^2(f)\|_{\Omega_T} h^2,$$

where $\widetilde{C}_k = (1 + \frac{k+1}{k}) \widetilde{C}_T$ is an absolute constant independent of f and h. □

Theorem 9. *Let $f \in C(\Omega^*)$ and $\mathcal{Q}_2(f)$ the quasi-interpolating spline for $k = 2$ defined in the previous section. Then the following statements hold.*

(i) If $f \in C^2(\Omega^*)$, then

$$||f - \mathcal{Q}_2(f)||_\Omega \leq C||D^2(f)||_{\Omega^*} h^2,$$

where $C > 0$ is an absolute constant independent of f and h.

(ii) If $f \in C^3(\Omega^*)$, then

$$||D^r(f - \mathcal{Q}_2(f))||_\Omega \leq \widetilde{C}||D^3(f)||_{\Omega^*} h^{3-r},\ r = 1, 2,$$

where $\widetilde{C} > 0$ is an absolute constant independent of f and h.

The complete proof follows closely the proof of Theorem 8 and can be found in [RK09].

5.5 Numerical Results

In this section, the approximation properties of our splines are demonstrated with numerical examples based on smooth test functions f. To do this, we use synthetic data sampled at the grid points $\mathbf{v} \in \mathcal{V}$ (see Section 5.3) to calculate the splines $s_f = \mathcal{Q}_k(f)$ for $k = \{2, 3\}$ on the partition $\Diamond$ of Ω. In our tests we consider three functions. The *blobby* function

$$\text{blob}(x, y, z) = \mathrm{e}^{-5((x-1/2)^2+y^2+z^2)} + \mathrm{e}^{-5(x^2+(y-1/2)^2+z^2)}$$

is a smooth blending of two metaballs [WMW86]. As a second representative for an exponential, we have the more complex function of *Franke* type [Fra79],

$$\begin{aligned}\text{fr}(x, y, z) =& 1/2\mathrm{e}^{-10((x-1/4)^2+(y-1/4)^2)} + 3/4\mathrm{e}^{-16((x-1/4)^2+(y-1/4)^2+(z-1/4)^2)} \\ &+1/2\mathrm{e}^{-10((x-3/4)^2+(y-1/8)^2+(z-1/2)^2)} - 1/4\mathrm{e}^{-20((x-3/4)^2+(y-3/4)^2)}.\end{aligned}$$

Finally, we provide the results for the well-known *Marschner Lobb* [ML94] function

$$\text{ml}(x, y, z) = \frac{1 - \sin(\pi z/2) + \alpha(1 + \cos(2\pi f_M \cos(\pi\sqrt{x^2 + y^2}/2)))}{2(1 + \alpha)},\ \alpha = \frac{1}{4}, f_M = 6,$$

which is highly oscillating and therefore a challenging test for any approximation method.

For each function we proceed with a series of tests on the cubic domain $\Omega^* = [-h, 1 + h]^3 \subseteq \mathbb{R}^3$ translated by $(-1/2, -1/2, -1/2)^\mathsf{T}$ with decreasing grid size h, which corresponds to a sampling on a finer partition. The first column in Tables 5.1 to 5.12 shows the increasing values of $1/h$ while the remaining columns show different errors for the function values (Tables 5.1 to 5.6) and the first and second derivatives in x-direction (Tables 5.7 to 5.12), respectively. The maximal error of the function values and the derivatives at the grid points is denoted as

$$\text{err}_{\text{data}} = \max\{|D_x^r(f(\mathbf{v}) - s_f(\mathbf{v}))| : \mathbf{v} \in \mathcal{V}\},\ r = 0, 1, 2,$$

$1/h$	$\mathrm{err}^{\mathrm{blob}}_{\mathrm{data}}$	$\mathrm{err}^{\mathrm{blob}}_{\mathrm{max}}$	$\mathrm{err}^{\mathrm{blob}}_{\mathrm{mean}}$	$\mathrm{err}^{\mathrm{blob}}_{\mathrm{rms}}$
8	0.072325	0.080944	0.011721	0.018593
16	0.018859	0.023713	0.003983	0.005837
32	0.004765	0.008360	0.001503	0.002074
64	0.001194	0.003504	0.000658	0.000892
128	0.000299	0.001614	0.000314	0.000426

Table 5.1: Approximation of blob by s_{blob} for $k = 3$.

$1/h$	$\mathrm{err}^{\mathrm{fr}}_{\mathrm{data}}$	$\mathrm{err}^{\mathrm{fr}}_{\mathrm{max}}$	$\mathrm{err}^{\mathrm{fr}}_{\mathrm{mean}}$	$\mathrm{err}^{\mathrm{fr}}_{\mathrm{rms}}$
8	0.200420	0.200536	0.010000	0.0220219
16	0.056525	0.065843	0.003392	0.0072288
32	0.014583	0.019627	0.001189	0.0023808
64	0.003685	0.007333	0.000481	0.0009176
128	0.000926	0.003185	0.000221	0.0004145

Table 5.2: Approximation of fr by s_{fr} for $k = 3$.

$1/h$	$\mathrm{err}^{\mathrm{ml}}_{\mathrm{data}}$	$\mathrm{err}^{\mathrm{ml}}_{\mathrm{max}}$	$\mathrm{err}^{\mathrm{ml}}_{\mathrm{mean}}$	$\mathrm{err}^{\mathrm{ml}}_{\mathrm{rms}}$
8	0.093599	0.165222	0.060747	0.071258
16	0.091361	0.102068	0.037767	0.045003
32	0.045369	0.048335	0.012682	0.015818
64	0.013786	0.015158	0.003474	0.004388
128	0.003309	0.004879	0.001003	0.001267

Table 5.3: Approximation of ml by s_{ml} for $k = 3$.

for a function f and $\mathrm{err}_{\mathrm{max}}$ is the (approximate) maximal error of s_f in the uniform norm on Ω. The latter error is computed on a fine discretization $\mathcal{X}$ of the domain by choosing a fixed number of 220 evenly distributed sample points in each tetrahedron:

$$\mathrm{err}_{\mathrm{max}} = \max\{|D_x^r(f(\mathbf{v}) - s_f(\mathbf{v}))| : \mathbf{v} \in \mathcal{X}\},\ r = 0, 1, 2.$$

For the sake of completeness, we also provide the mean error $\mathrm{err}_{\mathrm{mean}}$

$$\mathrm{err}_{\mathrm{mean}} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{v}\in\mathcal{X}} |D_x^r(f(\mathbf{v}) - s_f(\mathbf{v}))|,\ r = 0, 1, 2,$$

as well as the root mean square error

$$\mathrm{err}_{\mathrm{rms}} = \sqrt{\frac{1}{|\mathcal{X}|} \sum_{\mathbf{v}\in\mathcal{X}} |D_x^r(f(\mathbf{v}) - s_f(\mathbf{v}))|^2},\ r = 0, 1, 2,$$

$1/h$	$\mathrm{err}_{\mathrm{data}}^{\mathrm{blob}}$	$\mathrm{err}_{\mathrm{max}}^{\mathrm{blob}}$	$\mathrm{err}_{\mathrm{mean}}^{\mathrm{blob}}$	$\mathrm{err}_{\mathrm{rms}}^{\mathrm{blob}}$
8	0.053447	0.053719	0.007873	0.013093
16	0.013724	0.013783	0.002221	0.003573
32	0.003454	0.003466	0.000585	0.000922
64	0.000865	0.000867	0.000149	0.000233
128	0.000216	0.000216	0.000037	0.000058

Table 5.4: Approximation of blob by s_{blob} for $k = 2$.

$1/h$	$\mathrm{err}_{\mathrm{data}}^{\mathrm{fr}}$	$\mathrm{err}_{\mathrm{max}}^{\mathrm{fr}}$	$\mathrm{err}_{\mathrm{mean}}^{\mathrm{fr}}$	$\mathrm{err}_{\mathrm{rms}}^{\mathrm{fr}}$
8	0.157341	0.157385	0.007047	0.016126
16	0.042986	0.043009	0.002138	0.004814
32	0.010986	0.010992	0.000595	0.001312
64	0.002761	0.002763	0.000157	0.000342
128	0.000691	0.000691	0.000040	0.000087

Table 5.5: Approximation of fr by s_{fr} for $k = 2$.

$1/h$	$\mathrm{err}_{\mathrm{data}}^{\mathrm{ml}}$	$\mathrm{err}_{\mathrm{max}}^{\mathrm{ml}}$	$\mathrm{err}_{\mathrm{mean}}^{\mathrm{ml}}$	$\mathrm{err}_{\mathrm{rms}}^{\mathrm{ml}}$
8	0.088397	0.165560	0.060171	0.071867
16	0.081199	0.099245	0.033299	0.040339
32	0.031214	0.034153	0.009694	0.012068
64	0.008584	0.008928	0.002415	0.003028
128	0.002004	0.002132	0.000595	0.000748

Table 5.6: Approximation of ml by s_{ml} for $k = 2$.

on the same fine discretization $\mathcal{X}$.

As can be seen from Tables 5.1, 5.2 and 5.3, the approximation errors $\mathrm{err}_{\mathrm{max}}$, $\mathrm{err}_{\mathrm{mean}}$ and $\mathrm{err}_{\mathrm{rms}}$ for $s_f = \mathcal{Q}_3(f)$ decrease by about a factor of two for each function as the grid size goes down from h to $h/2$, while the error at the data points decreases by a factor of four. This confirms the results from Theorem 8, Section 5.4. Tables 5.4 to 5.6 show the approximation errors for $s_f = \mathcal{Q}_2(f)$, while Tables 5.7 to 5.12 show the approximation errors of the first and second derivatives in x-direction $D_x(f)$ and $D_{xx}(f)$, respectively. Here, the results from Theorem 9 are confirmed, since the error of the function values and the first derivatives decreases by a factor of four as the grid size goes down from h to $h/2$ and the error of the second derivatives decreases by a factor of two. This confirms our result that the operator $\mathcal{Q}_2$ approximates the derivatives of a sufficiently smooth function with optimal approximation order. For visual comprehension we provide the graphs for the errors of the function values (see Table 5.6) and first derivatives (Table 5.9) of the Marschner-Lobb function on a double logarithmic scale in Figure 5.5. In addition,

$1/h$	$\mathrm{err}_{\mathrm{data}}^{D_x(\mathrm{blob})}$	$\mathrm{err}_{\max}^{D_x(\mathrm{blob})}$	$\mathrm{err}_{\mathrm{mean}}^{D_x(\mathrm{blob})}$	$\mathrm{err}_{\mathrm{rms}}^{D_x(\mathrm{blob})}$
8	0.158589	0.241703	0.024732	0.038537
16	0.041100	0.071218	0.007084	0.010642
32	0.010475	0.018889	0.001888	0.002780
64	0.002647	0.004782	0.000487	0.000709
128	0.000759	0.001271	0.000124	0.000180

Table 5.7: Approximation of $D_x(\mathrm{blob})$ by $D_x(s_{\mathrm{blob}})$ for $k = 2$.

$1/h$	$\mathrm{err}_{\mathrm{data}}^{D_x(\mathrm{fr})}$	$\mathrm{err}_{\max}^{D_x(\mathrm{fr})}$	$\mathrm{err}_{\mathrm{mean}}^{D_x(\mathrm{fr})}$	$\mathrm{err}_{\mathrm{rms}}^{D_x(\mathrm{fr})}$
8	0.738493	0.916685	0.042349	0.094669
16	0.210754	0.300654	0.013332	0.029001
32	0.057464	0.082984	0.003703	0.007914
64	0.014565	0.021206	0.000971	0.002056
128	0.003674	0.005374	0.000248	0.000523

Table 5.8: Approximation of $D_x(\mathrm{fr})$ by $D_x(s_{\mathrm{fr}})$ for $k = 2$.

$1/h$	$\mathrm{err}_{\mathrm{data}}^{D_x(\mathrm{ml})}$	$\mathrm{err}_{\max}^{D_x(\mathrm{ml})}$	$\mathrm{err}_{\mathrm{mean}}^{D_x(\mathrm{ml})}$	$\mathrm{err}_{\mathrm{rms}}^{D_x(\mathrm{ml})}$
8	5.03099	5.33248	1.575060	2.042270
16	4.21588	4.58655	0.887093	1.254340
32	1.52074	2.78363	0.254649	0.382565
64	0.42781	0.80967	0.063386	0.097711
128	0.10828	0.21132	0.015481	0.024102

Table 5.9: Approximation of $D_x(\mathrm{ml})$ by $D_x(s_{\mathrm{ml}})$ for $k = 2$.

Figure 5.6 visualizes the reconstruction error for the Marschner-Lobb function sampled from a sparse grid (64^3 data points) using the type-6 splines (quadratic super splines and S_3^1 splines, respectively), as well as our new operator $\mathcal{Q}_2$. It can be seen from the figure that our $\mathcal{Q}_2$ spline reconstructs the Marschner-Lobb function with a lower error.

In all cases, the computation of the splines s_f, even on the finest partition of Ω involving more than two million data samples, were computed in less than one minute on a standard PC, depending on the complexity of f. In the numerical tests, several techniques for fast evaluation of s_f can be employed, e.g. de Casteljau's algorithm, which provides the functional value and the derivatives simultaneously, see Section 3.4.1.

$1/h$	$\mathrm{err}_{\mathrm{data}}^{D_{xx}(\mathrm{blob})}$	$\mathrm{err}_{\mathrm{max}}^{D_{xx}(\mathrm{blob})}$	$\mathrm{err}_{\mathrm{mean}}^{D_{xx}(\mathrm{blob})}$	$\mathrm{err}_{\mathrm{rms}}^{D_{xx}(\mathrm{blob})}$
8	2.78180	7.36277	0.647459	0.944675
16	1.51370	4.48919	0.366841	0.525985
32	0.78379	2.34272	0.192558	0.273797
64	0.39897	1.18080	0.098316	0.139153
128	0.22540	0.61108	0.049920	0.070501

Table 5.10: Approximation of $D_{xx}(\mathrm{blob})$ by $D_{xx}(s_{\mathrm{blob}})$ for $k = 2$.

$1/h$	$\mathrm{err}_{\mathrm{data}}^{D_{xx}(\mathrm{fr})}$	$\mathrm{err}_{\mathrm{max}}^{D_{xx}(\mathrm{fr})}$	$\mathrm{err}_{\mathrm{mean}}^{D_{xx}(\mathrm{fr})}$	$\mathrm{err}_{\mathrm{rms}}^{D_{xx}(\mathrm{fr})}$
8	9.48307	34.2698	1.41213	2.72682
16	6.56458	24.2788	0.87606	1.68534
32	3.35031	13.2468	0.48023	0.91651
64	1.63083	6.82766	0.24989	0.47418
128	0.81025	3.45017	0.12726	0.24070

Table 5.11: Approximation of $D_{xx}(\mathrm{fr})$ by $D_{xx}(s_{\mathrm{fr}})$ for $k = 2$.

$1/h$	$\mathrm{err}_{\mathrm{data}}^{D_{xx}(\mathrm{ml})}$	$\mathrm{err}_{\mathrm{max}}^{D_{xx}(\mathrm{ml})}$	$\mathrm{err}_{\mathrm{mean}}^{D_{xx}(\mathrm{ml})}$	$\mathrm{err}_{\mathrm{rms}}^{D_{xx}(\mathrm{ml})}$
8	223.618	258.992	55.6158	77.3912
16	147.363	314.767	43.9654	66.5142
32	63.4782	306.433	25.2862	42.1393
64	29.2791	209.945	12.7598	22.2219
128	13.8998	118.557	6.2608	11.0415

Table 5.12: Approximation of $D_{xx}(\mathrm{ml})$ by $D_{xx}(s_{\mathrm{ml}})$ for $k = 2$.

5.6 Conclusion and Remarks

The BCC lattice has gained increased popularity in numerous applications, e.g. volume rendering [Cse05], skeletonization [BB08], and topology preserving reconstruction [SS06]. It is well known that the BCC lattice is a superior sampling grid compared to the cubic lattice and therefore equal approximation quality can be achieved with fewer data samples. The nature of C^1-splines on truncated octahedral partitions motivates the direct usage of data on the BCC lattice. It is possible to adapt the presented approximation scheme to such data sets. In this context, we are confident to achieve even better approximation properties, in particular an optimal order of approximation.

In a next step, the structure of C^2-splines on truncated octahedral partitions should be analyzed. First investigations on this topic have shown that local quasi-interpolation operators with an expected degree of four could be constructed. To our knowledge, this

would be the first C^2-smooth spline approximation scheme with a lower degree than five.

Remark 1. *The quasi-interpolation scheme for C^1-splines on truncated octahedral partitions described in Section 5.3 can be easily extended to a cubic domain by the usage of half, quarter and eighth truncated octahedra. Thereby, no additional data points are needed. However, to guarantee the approximation order of the splines at the boundary we have to use data points outside of the domain (see Section 5.3). But these additional data points needed can also be constructed by local extrapolation at the boundary of the given data such that the approximating spline covers the whole initial data volume.*

Remark 2. *It is possible to modify our construction of an approximating spline to an interpolating spline with a simple trick. We assume that the data lie on a cubic grid with grid-size h. Then we construct a truncated octahedral partition with size h, where half of the truncated octahedra are centered at the data points. In these truncated octahedra we apply the data value in the center to the eight data points on the hexagonal faces, needed for our quasi-interpolation scheme. Therefore, we have constructed all data points needed for the calculation of the spline on the whole partition and the resulting spline interpolates the initial data values. However, we lose the approximation properties by this approach.*

Remark 3. *We have also applied our scheme for $k = 2$ to the simple test functions*

$$f_1(x, y, z) = xy + xz + yz + x + y + z + 1 \text{ and } f_2(x, y, z) = x^2$$

in order to verify the results of Lemma 4 and Corollary 5. To be more precise, the errors of $s_{f_1} = \mathcal{Q}_2(f_1)$ are (numerical) zero for the functional values and the derivatives, confirming Lemma 4 (i), (ii) and (iii). The error of $s_{f_2} = \mathcal{Q}_2(f_2)$ is constantly $h^2/4$, in accordance with Lemma 4, (iv) (and (v)), while the errors for the corresponding derivatives are zero, which confirms Corollary 5. For the trilinear function

$$f_3(x, y, z) = axyz$$

our tests show that the errors of $s_{f_3} = \mathcal{Q}_2(f_3)$ and $D_x(s_{f_3})$ decrease by a factor of 8 for $h \to h/2$, meaning that in this case, we have the optimal approximation order three.

We conclude this chapter with a first visualization of our spline $\mathcal{Q}_2$ in Figure 5.7. Here, we apply ray casting to reconstruct Barth's sextic function $4(\tau^2x^2 - y^2)(\tau^2y^2 - z^2)(\tau^2z^2 - x^2) - (1 + 2\tau)(x^2 + y^2 + z^2 - w^2)^2w^2)$, $\tau = \frac{1}{2}(1 + \sqrt{(}5))$, sampled from a 128^3 grid. Further results can be found in [MKRG11] and Chapter 6.

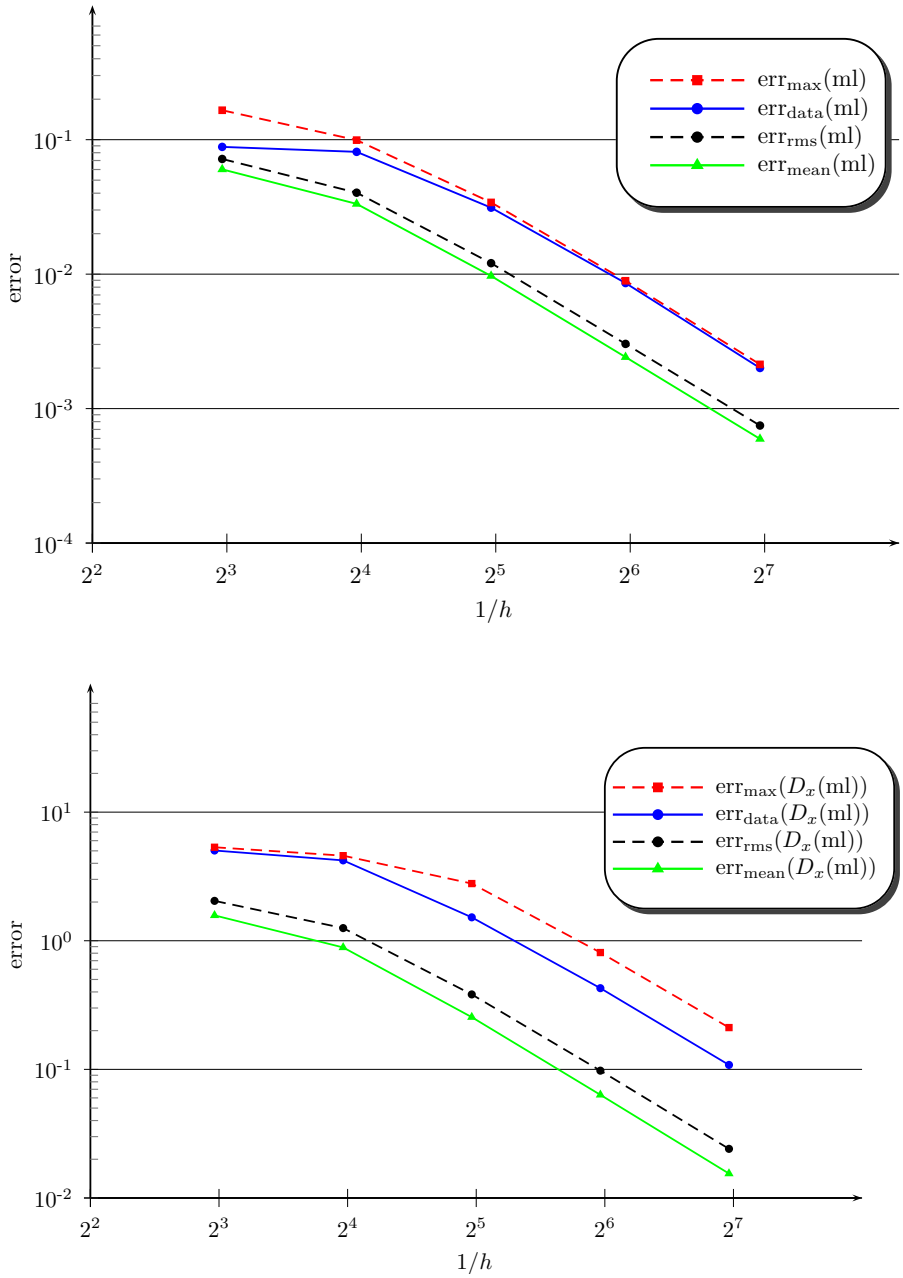

Figure 5.5: *Top*: graphs of the approximation errors of the Marschner-Lobb function ml by s_{ml} for $k = 2$. on a double logarithmic scale. *Bottom*: graphs of the approximation errors of the derivatives $D_x(\mathrm{ml})$ by $D_x(s_{\mathrm{ml}})$ for $k = 2$.

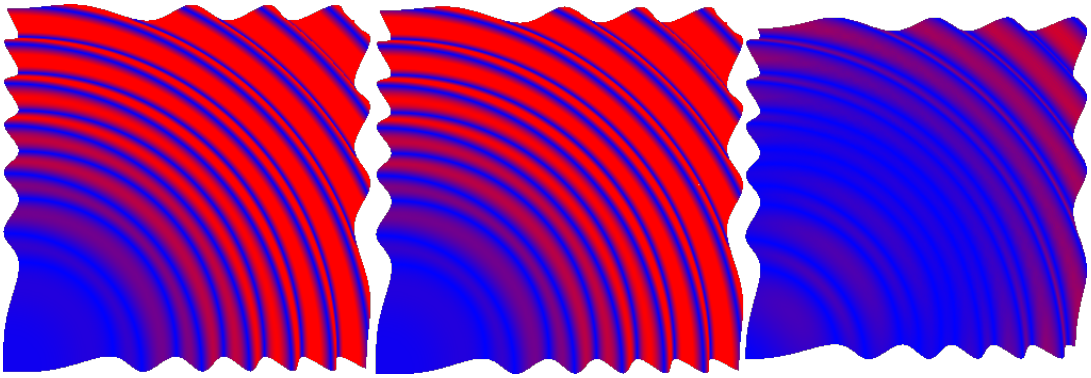

Figure 5.6: Error visualization of the Marschner-Lobb function sampled from a 64^3 grid. Red indicates high error, blue means low error. *Left*: quadratic super splines. *Middle*: S_3^1 spline. *Right*: quasi-interpolating operator $\mathcal{Q}_2$.

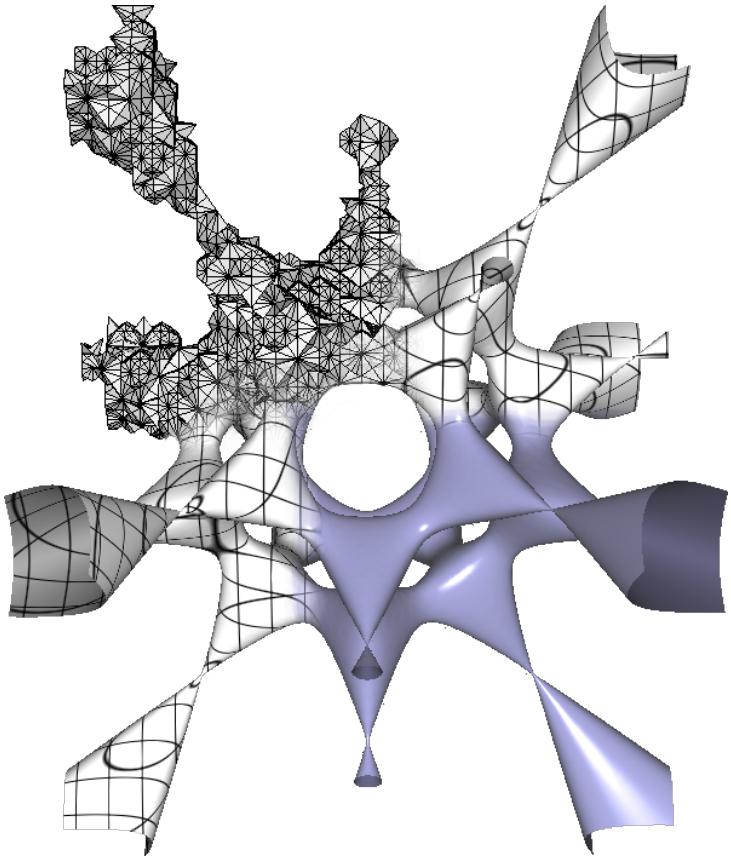

Figure 5.7: Visualization of the Barth sextic function sampled from a 128^3 grid. Top left we can see the tetrahedra and some truncated octahedra on the surface. In the middle we see quadratic boundary curves along constant cutting planes. The smooth surface with Phong shading applied can be seen on the bottom right.

Chapter 6

GPU Kernels for Piecewise Quadratic and Cubic Approximation

In this chapter, we give the details of our GPU kernels for hardware-accelerated visualization of surfaces from volume data based on trivariate splines. As our main contribution, these are the first GPU implementations for trivariate splines taking their structure into account such that bus traffic, memory consumption, and data access on the GPU is optimized. In comparison to former pure CPU implementations [RZNS03] and GPU implementations [SGS06] based on quadratic super splines we achieve a speedup of two to three orders of magnitude. We further address the more complex cases of cubic C^1-splines, see Section 4.4, and quadratic C^1-splines on truncated octahedral partitions, see Chapter 5. In addition, our schemes for efficient geometry encoding and on-the-fly computation of spline coefficients allow us for the first time to apply trivariate splines for real-time high-quality visualizations of surfaces from very large volume data sets arising in practical applications (i.e., medical or scientific visualization).

The approximative nature of the spline algorithms and the fact that we choose splines in B-form of low and lowest possible degree provide several advantages for our GPU implementations. Using *blossoming* (see Section 3.4.3), the restrictions of the splines along viewing rays are easily determined in the fragment shader by applying a few *de Casteljau* steps (see Section 3.4.1). This is connected with an efficient clipping procedure which does not require explicit knowledge on the vertices of the current tetrahedron and avoids clipping with its four triangular planes. The low polynomial degrees support analytic and fast iterative root finding algorithms to solve the quadratic and cubic equations, respectively, which we can perform in parallel on the GPU with a low number of fixed iterations in the latter case. Moreover, the computation of the gradients necessary for Phong illumination is done by applying *Bernstein-Bézier techniques* to the polynomial pieces and a separate model for the approximative derivatives is not needed. The *convex hull property* of the B-form allows us to quickly determine if a given polynomial on a tetrahedron T does not contribute to the visible surface (i.e., all spline coefficients on T are either below or above the isolevel). In this case, these *non-relevant* tetrahedra can be excluded from further costly processings.

We first describe a hybrid cell projection / ray casting approach in Section 6.1 which has been proven to be well suited for rendering of surfaces from small to medium sized data sets (say, up to 256^3 voxels) and GPUs with lower performance, e.g., mobile laptop GPUs. This has led to an entire image based volume ray casting approach which is better

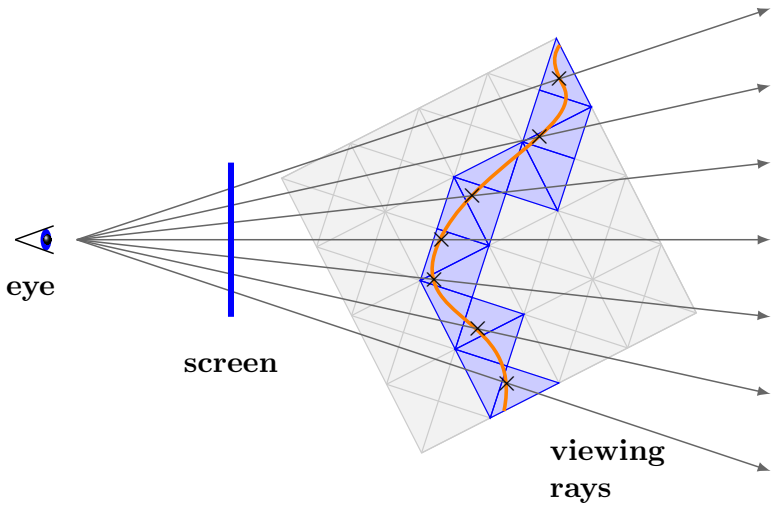

Figure 6.1: A 2D illustration of the cell projection / ray casting principle for isosurface visualization. On each triangle satisfying the convex hull property (blue), the corresponding Bézier curve is intersected with the rays emerging from the viewer's position.

suited for very large volume data sets, see Section 6.2. The results involving techniques from Computer Graphics illustrate that our real-time reconstruction and visualization based on trivariate quasi-interpolating splines satisfies the various requirements of high-quality visualization. We show visual results and give a comparative analysis of our kernels in Section 6.3.

6.1 Hybrid Cell Projection / Ray Casting

Our algorithm can be divided into two main parts: first, we do a preprocessing for surface reconstruction where we determine the relevant or *active* tetrahedra which do contribute to the final surface. In addition, the data structures for the bounding tetrahedra are prepared. Our first implementations [KZ08] relied on a CPU preprocessing which along with the data transfer to the GPU takes a couple of seconds and has to be repeated for each change of isolevel or data set. However, in many applications it is crucial to vary the isosurface interactively in order to gain deeper understanding of the data to be visualized. In addition, the data set itself may vary, for instance in numerical simulations. Therefore, in [KKG09] we accelerate this preprocess significantly using a flexible programming model on modern GPUs for high performance data parallel computing. We describe our preprocessing in Section 6.1.1.

The second part uses vertex and fragment programs for the visualization of the surface in a combined rasterization / ray casting approach, see Section 6.1.2. A 2D illustration of the visualization principle is shown in Figure 6.1. For each active tetrahedron the bounding geometry is processed in the graphics pipeline. The vertex programs initialize various parameters, such as the viewing rays, and appropriate barycentric coordinates.

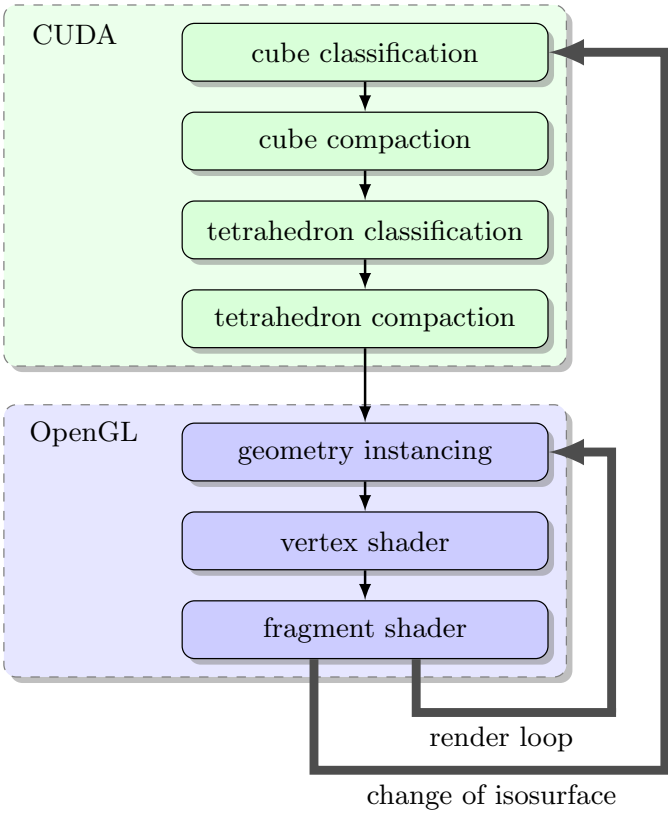

Figure 6.2: Our GPU algorithm is organized as a combined CUDA (shaded green) and OpenGL (shaded blue) approach. The CUDA part is responsible for specifying the relevant cubes and tetrahedra, respectively, when the isovalue or data set is modified. The visualization is performed in the OpenGL rendering pipeline.

The fragment programs then test for ray-patch intersections. A schematic overview of our GPU reconstruction / visualization pipeline is shown in Figure 6.2.

Compared with standard models for volume visualization, such as trilinear interpolation based on a simple cube partition, trivariate splines have a much more complex structure. In order to keep bus traffic, memory consumption, and data access on the GPU as low as possible, data streams have to be organized appropriately. We therefore develop an implicit scheme for the encoding of the bounding geometry of the relevant tetrahedra. Our implicit scheme exploits the uniform structure of the underlying tetrahedral partitions to prevent explicit usage and storage of the geometry. This is of major importance for efficient visualizations based on trivariate splines, since otherwise, the storage required for the bounding geometry of real-world data sets exceeds the available GPU memory. In order to encode the necessary bounding geometries, i.e., the active tetrahedra, we construct a triangle strip for each of the 24 *generic tetrahedra* within the unit cube for the splines based on the type-6 partition Δ_6 and accordingly, we construct a triangle strip for each of the 144 tetrahedra in one truncated octahedron for the

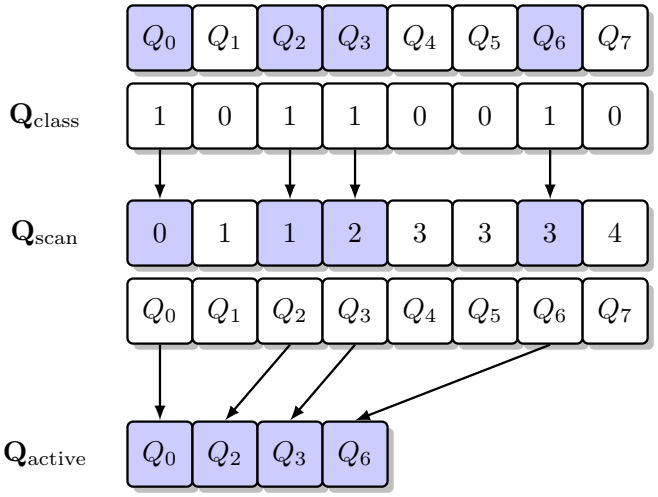

Figure 6.3: Illustration of parallel stream compaction using prefix scans. For each entry with a 1 in $\mathbf{Q}_{\text{class}}$, $\mathbf{Q}_{\text{scan}}$ contains a unique address into the compressed array $\mathbf{Q}_{\text{active}}$. The sum of the last entries of $\mathbf{Q}_{\text{class}}$ and $\mathbf{Q}_{\text{scan}}$ gives the size of $\mathbf{Q}_{\text{active}}$.

quadratic C^1-spline. These generic triangle strips are stored as separate vertex buffer objects (VBOs) on the GPU. Each VBO is then used to draw all the active tetrahedra of its type by using *instancing*.

6.1.1 Preprocessing for Efficient Visualization

The main purpose of our preprocessing is to determine the tetrahedra which are relevant for visualization. We therefore develop suitable GPU kernels based on CUDA to allow for an interactive change of isosurface. We first start a kernel thread for each cube Q in the type-6 partition Δ_6 that computes the coefficients of the determining set $\mathcal{M} \subseteq \mathcal{D}_q(\Delta_6)$ from which the remaining coefficients can be found quickly using simple averaging. The *cube classification* tests if all coefficients of $\mathcal{M}$ are either below or above the isolevel. In this case, it follows from the convex hull property of the B-form that the polynomials in Q cannot contribute to the surface and we can exclude Q from further examination. Otherwise, Q contains at least one tetrahedron with a visible surface patch. The result of the classification is written in the corresponding entry of a linear integer array $\mathbf{Q}_{\text{class}}$ of size $(n+1)^3$: we write a 1 in $\mathbf{Q}_{\text{class}}$ if Q passes the classification test and a 0 otherwise. From this unsorted array $\mathbf{Q}_{\text{class}}$ we construct a second array $\mathbf{Q}_{\text{scan}}$ of the same size using the parallel prefix scan from the *CUDA data parallel primitives* (CUDPP) library. For each *active* cube, i.e., cubes with a 1 in $\mathbf{Q}_{\text{class}}$, $\mathbf{Q}_{\text{scan}}$ then contains a unique index corresponding to the memory position in a compacted array $\mathbf{Q}_{\text{active}}$. Since we use an exclusive prefix scan, the sum of the last entries of $\mathbf{Q}_{\text{class}}$ and $\mathbf{Q}_{\text{scan}}$ corresponds to the number of active cubes a for the surface. In the *compaction* step, we reserve memory for the array $\mathbf{Q}_{\text{active}}$ of size a. For each active Q_{ijk}, we write the index $i+j\cdot(n+1)+k\cdot(n+1)^2$ in $\mathbf{Q}_{\text{active}}$ at the position given by the corresponding entry of $\mathbf{Q}_{\text{scan}}$. An illustration of

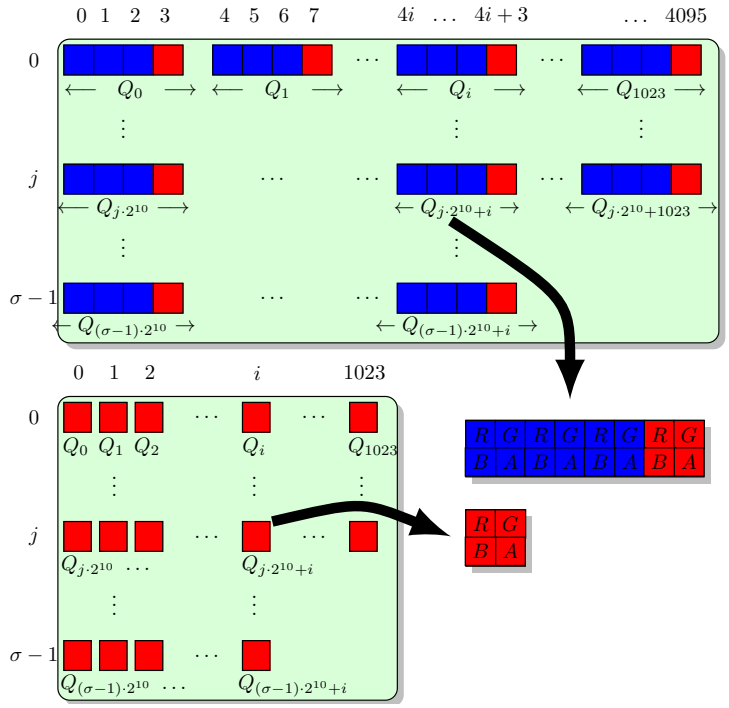

Figure 6.4: The layout of the textures used for the storage of the determining sets $\mathcal{M}_Q$ for quadratic super splines on the type-6 partition. Colors indicate the correspondence with Figure 4.3.

the parallel prefix scan is shown in Figure 6.3.

Similarly, we compact the active tetrahedra. Since we use instancing for later rendering of the tetrahedra, we reserve 24 arrays $\mathbf{T}_{\text{class},i}$, where each $\mathbf{T}_{\text{class},i}$ has size a and corresponds to one of the 24 different orientations of the generic tetrahedra in the type-6 partition. For each active cube Q, a kernel thread performs the classification for $T_0, T_1, \ldots, T_{23} \in Q$, now using the convex hull property of the B-form on the tetrahedra. Note that the B-coefficients for Q have to be calculated only once and are then assigned to the corresponding domain points on the tetrahedra using a constant lookup table. The following stream compaction works exactly as described above, except that here we use 24 distinct arrays $\mathbf{T}_{\text{scan},i}$, and the compaction is done by a set of 24 kernels (one for each type of tetrahedron) which write their results into the arrays $\mathbf{T}_{\text{active},i}$. These arrays are interpreted as pixel buffer objects, which can be directly used to render the bounding geometry of the piecewise polynomials. The same procedure applies for quadratic splines on truncated octahedral partitions except that on each truncated octahedron we have 144 generic tetrahedra.

The memory requirements for the GPU preprocessing are determined by the size of the data set as well as the number of active cubes, respectively tetrahedra, for a particular isovalue. In the first classification step we thus need to store the data set plus two (temporary) integer arrays of the same size for $\mathbf{Q}_{\text{class}}$ and $\mathbf{Q}_{\text{scan}}$, and the compacted array

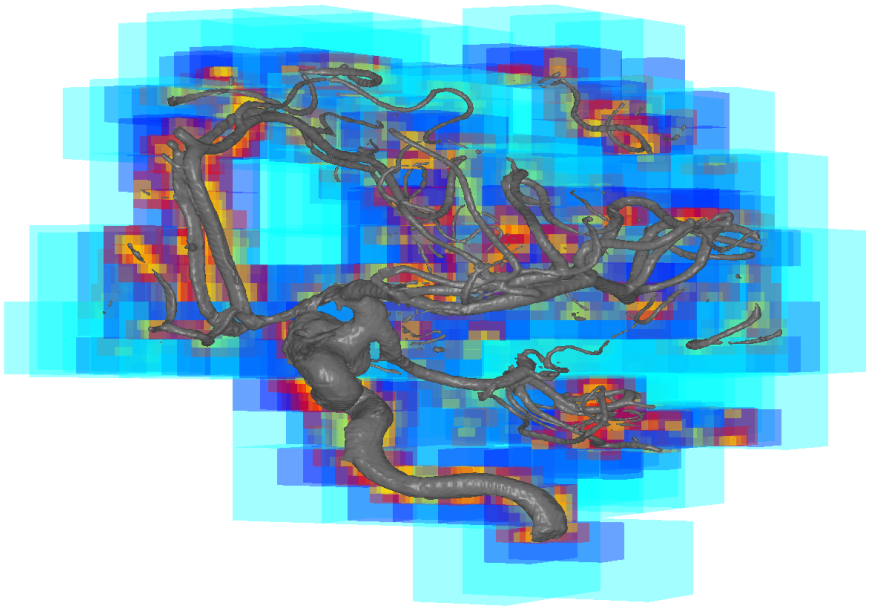

Figure 6.5: A spline isosurface and its min-max octree of the quadratic super spline approximating aneurism data. Colors indicate different node levels.

$\mathbf{Q}_{\text{active}}$. In the tetrahedron classifications we need space for $\mathbf{Q}_{\text{active}}$ and the (temporary) arrays $\mathbf{T}_{\text{class,i}}$, $\mathbf{T}_{\text{scan,i}}$ of the same size as well as the arrays $\mathbf{T}_{\text{active,i}}$. In our tests, we were able to reconstruct isosurfaces consisting of several million relevant tetrahedra and reconstruction times are in the range of 200 ms for quadratic splines, and 300 ms for cubic splines, respectively, see also Section 6.1.3.

Optionally, we can also store the precomputed coefficients of the determining set $\mathcal{M}_Q$ for each active Q on the GPU. The remaining coefficients can then be quickly computed by repeated averaging during visualization. This leads to increasing frame rates at the cost of higher memory usage, see Section 6.1.3. Figure 6.4 exemplarily shows the encoding scheme for quadratic splines on the type-6 partition. Since 20 B-coefficients have to be stored for each cube we use two textures in this case. We choose the first texture to consist of the 12 "blue" and 4 "red" coefficients associated with the domain points from the determining set in four successive RGBA quadruplets. The second texture contains the 4 remaining "red" coefficients. In the case of cubic splines 56 coefficients have to be encoded for each cube where we use four textures. Our tests have shown that for each active cube on average about twelve tetrahedra are relevant. Therefore, compared to direct approaches where the B-coefficients of all polynomial pieces are sent to the GPU our encoding decreases memory requirements by about a factor of 4 to 6. We emphasize that each row of the textures consists of 1024 entries corresponding with current hardware architectures where 4096 is the usual upper bound for the maximum width and height of 2D textures. Although we use only two and four

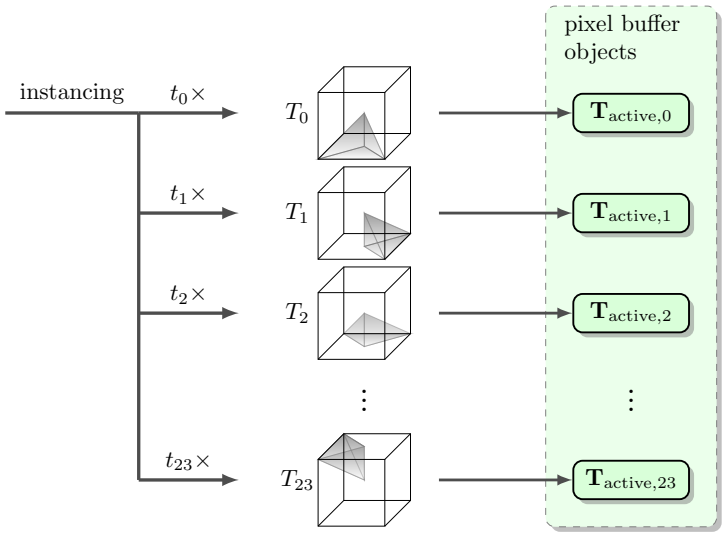

Figure 6.6: The geometry of each generic tetrahedron within the type-6 partition is stored only once. For rendering we use the instance id as an index into the arrays $\mathbf{T}_{\text{active},i}$ to obtain the world coordinates of the tetrahedra.

textures, respectively, this allows to store huge spline models consisting of up to 50 million polynomials.

On machines with limited GPU resources we can do the preprocessing on the CPU. In this case, we do not need to use prefix scans to compress the results obtained from classification but we use spatial data structures such as *min-max octrees* in order to increase efficiency. Every node in the octree is characterized by its minimal and maximal data values $f_{\min}, f_{\max}$ of its successors. Fixing an arbitrary isovalue ρ, only branches of the octree with $\rho \in [f_{\min}, f_{\max}]$ have to be considered. An isosurface within its associated min-max octree is shown in Figure 6.5. For the remaining data we proceed by exploiting the convex hull property of the B-form to indicate all tetrahedra which possibly contribute to the final surface. In practice, we observe that these preprocessing steps essentially reduce the amount of data (i.e. B-coefficients, bounding geometry) to be sent to or processed by the GPU. For large isosurfaces consisting of several million tetrahedra timings of the CPU preprocessing plus the data transfer to the GPU are in the range of 10 seconds.

6.1.2 GPU Visualization Details

The preprocessing from the previous section provides us with the arrays $\mathbf{T}_{\text{active},i}$ (and optionally the textures with precomputed coefficients from the determining sets) which can be directly used as an efficient description of the bounding tetrahedra. The processing of each of these arrays in the graphics pipeline is initiated with a single API call where the number of tetrahedra is given by the size of $\mathbf{T}_{\text{active},i}$, see Figure 6.6. In the following

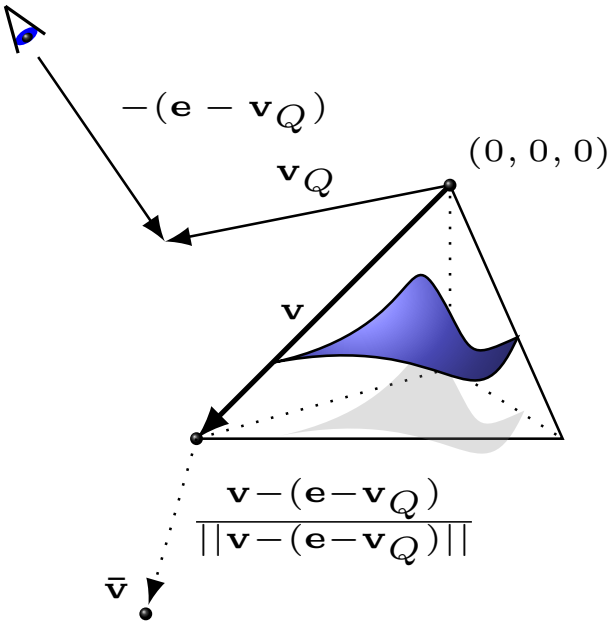

Figure 6.7: Companion point $\bar{\mathbf{v}}$ for tetrahedron vertex $\mathbf{v}$ and center point $\mathbf{v}_Q$.

we describe the details of our vertex and fragment programs where we use 24 different shader sets for the splines on the type-6 partition, and 144 different sets for the spline on the truncated octahedral partition. Since each tetrahedron has its dedicated vertex and fragment programs we can almost completely avoid conditional branches in the shaders. This is in compliance with current GPU architectures which are most efficient if shader computations do not diverge.

Vertex Shader Computations

For every vertex $\mathbf{v}_\mu, \mu = 0,1,2,3$, of a tetrahedron T, the vertex program first determines T's displacement $\mathbf{v}_Q$ from a texture reference into $\mathbf{T}_{\text{active},i}$ using the provided instance id. For later computation of the ray-patch intersection in the fragment shader we find the barycentric coordinates $\phi_{\mu,\nu}$, $\nu = 0,1,2,3$ as $\phi_{\mu,\nu}(\mathbf{v}_\mu) = \delta_{\mu,\nu}$. In addition, we determine the *companion point* $\bar{\mathbf{v}}$ on the viewing ray corresponding to the unit length extension of the vector defined by $\mathbf{v}_\mu + \mathbf{v}_Q$ and the eye point $\mathbf{e}$, see Figure 6.7. The companion point is used for efficient clipping of intersections with the tetrahedron T during fragment processing. The barycentric coordinates of the companion point, $\bar{\phi}_{\mu,\nu}(\bar{\mathbf{v}}) = \bar{\phi}_{\mu,\nu}(\mathbf{v}_\mu + (\mathbf{v}_\mu - (\mathbf{e}-\mathbf{v}_Q))/||\mathbf{v}_\mu - (\mathbf{e}-\mathbf{v}_Q)||)$, are computed according to Section 3.3.1 where the matrices M_T^{-1} are precomputed once for each generic tetrahedron. The barycentric coordinates $\phi_{\mu,\nu}, \bar{\phi}_{\mu,\nu}$ are then interpolated across T's triangular faces during rasterization.

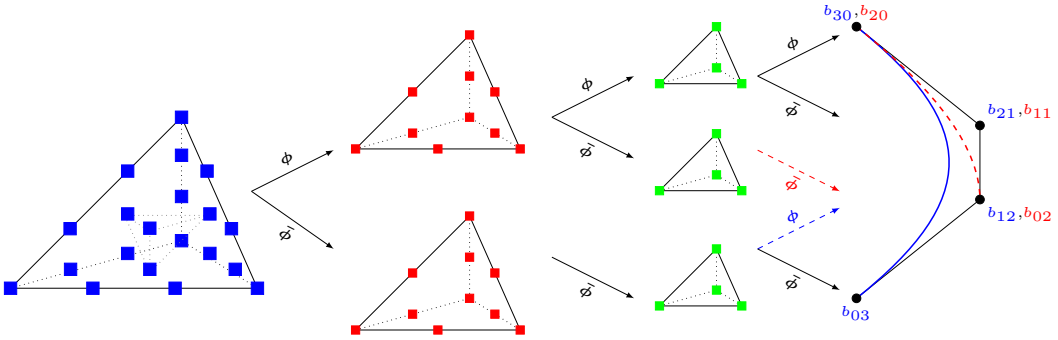

Figure 6.8: Trivariate blossoming in $\mathcal{P}_q$, $q = 2, 3$, provides us with the B-coefficients of a functional Bézier curve.

Fragment Shader Computations

For every fragment of the front-facing triangles of the bounding tetrahedra the fragment program test for intersections of the viewing ray with the associated polynomial $p = s|_T$. To do this, we first have to determine the 10 B-coefficients on p for quadratic splines, and 20 coefficients in the cubic case, respectively. We read the data values $f(\mathbf{v}_Q)$ and its neighbors from the volume texture for on-the-fly computation of the polynomial coefficients. For the cubic C^1-spline 23 neighboring values need to be fetched. In the case of quadratic super splines we need the complete 27 neighborhood since some smoothness conditions are averaged. For splines on the truncated octahedral partition 28 neighbors are needed. We can then obtain the $b_{ijk\ell}$ according to Section 4.4 or 5.3. Alternatively, we can use the precomputed determining sets for each Q stored as textures, see Section 6.1.1, corresponding to five texture fetches for quadratic super splines, and fourteen fetches for cubics, respectively. Then, only the remaining coefficients of p need to be computed by simple repeated averaging. This method is less memory efficient but leads to a slightly improved rendering performance. For a comparison see Section 6.1.3.

In the actual ray-surface intersection tests we need a univariate representation of p restricted along the viewing ray. Using trivariate blossoming, we obtain a functional Bézier curve for a simplified root finding. In addition, we can re-use intermediate results from the blossoms for quick gradient calculation and do not need to explicitly determine the exit point of the ray w.r.t. T for clipping of intersections against the bounding geometry. Using the interpolated barycentric coordinates $\boldsymbol{\phi} = (\phi_0, \phi_1, \phi_2, \phi_3)$ and $\bar{\boldsymbol{\phi}} = (\bar{\phi}_0, \bar{\phi}_1, \bar{\phi}_2, \bar{\phi}_3)$ obtained from rasterization, we can read off the B-coefficients of the functional curve directly from the blossoms. For instance, in the cubic case we set

$$b_{30} = p[\boldsymbol{\phi}, \boldsymbol{\phi}, \boldsymbol{\phi}],\ b_{21} = p[\boldsymbol{\phi}, \boldsymbol{\phi}, \bar{\boldsymbol{\phi}}],\ b_{12} = p[\bar{\boldsymbol{\phi}}, \bar{\boldsymbol{\phi}}, \boldsymbol{\phi}],\ \text{and}\ b_{03} = p[\bar{\boldsymbol{\phi}}, \bar{\boldsymbol{\phi}}, \bar{\boldsymbol{\phi}}],$$

see Figure 6.8. Since intermediate results can be re-used the first step of de Casteljau's algorithm, which accounts for ten inner products each, has to be performed for $\boldsymbol{\phi}$ and $\bar{\boldsymbol{\phi}}$ only once. We proceed in the same way for the second de Casteljau step with $b^{[1]}_{ijk\ell}(\boldsymbol{\phi})$ (resulting from the first step with $\boldsymbol{\phi}$) and $\boldsymbol{\phi}$, as well as $b^{[1]}_{ijk\ell}(\bar{\boldsymbol{\phi}})$ and $\bar{\boldsymbol{\phi}}$, where $i+j+k+\ell =$

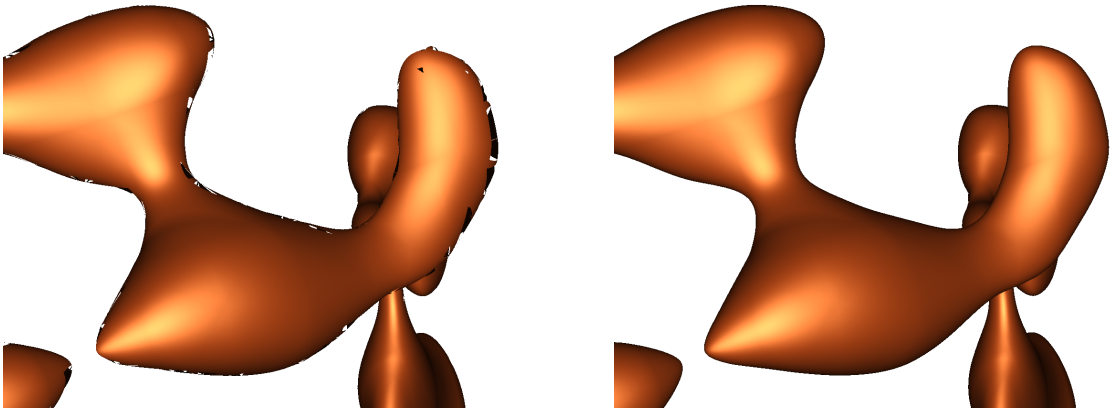

Figure 6.9: The effect of an insufficient number of iterations for root finding of cubic polynomials. *Left*: three iterations lead to artifacts in particular near silhouettes. *Right*: five iterations are sufficient to obtain precise intersections.

2, using in total eight inner products. Finally, the blossoms are completed with four additional inner products using $b^{[2]}_{ijk\ell}(\boldsymbol{\phi})$ and $b^{[2]}_{ijk\ell}(\bar{\boldsymbol{\phi}})$, $i+j+k+\ell=1$, which correspond to the remaining de Casteljau steps on the last level. For quadratic splines this scheme simplifies to

$$b_{20} = p[\boldsymbol{\phi}, \boldsymbol{\phi}],\ b_{11} = p[\boldsymbol{\phi}, \bar{\boldsymbol{\phi}}],\ b_{02} = p[\bar{\boldsymbol{\phi}}, \bar{\boldsymbol{\phi}}].$$

Next, the monomial form of the Bézier curve, $\sum_{i=0}^{q} x_i \cdot t^i$, is solved for the ray parameter t. In case of quadratic splines we can use any numerical stable scheme to find the roots $t^{(0)}, t^{(1)}$ of a quadratic polynomial, see also [Sch90]. There are several ways to obtain the roots of a cubic equation [Sch90, HE95], for example using Cardano's formula, or applying a recursive Bézier hull subdivision algorithm [SD07]. Since the first method involves trigonometric functions and the second does not converge very quickly, similarly to the bivariate setting in [RZHB*08] we opt for an iterative Newton approach. As starting values we choose $t_1^{(0)} = -x_0/x_1, t_1^{(1)} = (1/4(x_3+x_2)-x_0)/(3/4 \cdot x_3 + x_2 + x_1)$ and $t_1^{(2)} = (2 \cdot x_3 + x_2 - x_0)/(3 \cdot x_3 + 2 \cdot x_2 + x_1)$. Note that this corresponds to the first Newton iteration starting with $0, 1/2$, and 1, respectively. Four additional iterations with

$$t_{j+1}^{(\mu)} = \frac{(t_j^{(\mu)})^2(2 \cdot t_j^{(\mu)} x_3 + x_2) - x_0}{t_j^{(\mu)}(3 \cdot t_j^{(\mu)} x_3 + 2 \cdot x_2) + x_1}, \quad \mu = 0, 1, 2,$$

suffice to find precise intersections without notable artifacts. The effect of insufficient iterations for root finding in the cubic case is shown in Figure 6.9.

For each valid solution $t \in t^{(\mu)}$, $\mu = 0, 1$, in the quadratic case, and $t_5^{(\mu)}$, $\mu = 0, 1, 2$, in the cubic case, respectively, the associated barycentrics $\boldsymbol{\phi}(\mathbf{x}(t))$ are found by a simple linear interpolation with $\boldsymbol{\phi}$ and $\bar{\boldsymbol{\phi}}$. We take the first zero t where all the components of $\boldsymbol{\phi}(\mathbf{x}(t))$ are positive, i.e., the intersection point lies inside T. If no such t exists we discard

the fragment. From the multi-affine property of the blossom (see Section 3.4.3) it follows that the directional derivatives $D_u p(\mathbf{x}(t))$, see Equation (3.17), are obtained by a linear interpolation of $b^{[1]}_{ijk\ell}(\boldsymbol{\phi})$ and $b^{[1]}_{ijk\ell}(\bar{\boldsymbol{\phi}})$, with the ray parameter t, which is in the cubic case followed by a de Casteljau step on the second level using $\boldsymbol{\phi}(\mathbf{x}(t))$. Finally, we calculate the gradient $\nabla p = (D_x p, D_y p, D_z p)^\mathsf{T}$ for later illumination according to Equation (3.16) with three additional scalar products. Here, the $\partial\phi_\mu/\partial x, \partial\phi_\mu/\partial y, \partial\phi_\mu/\partial z$, $\mu = 0, 1, 2, 3$, are precomputed once for each of the generic tetrahedra.

For clarification, we show the simplified pseudo-code of our fragment program for cubic splines in Listing 6.1.

```
/* interpolated barycentric coordinates */
in vec4 bcIn;       /* φ(v) */
in vec4 bcOut;      /* φ(v̄) */

flat in  vec3 position;  /* index into volume texture */
uniform  sampler3D volumeTexture; /* volume data set */
uniform  float isovalue;
/* derivatives of the barycentric coordinates */
const vec4 dphidx;       /* ∂φ_μ/∂x, μ=0,1,2,3 */
const vec4 dphidy;       /* ∂φ_μ/∂y, μ=0,1,2,3 */
const vec4 dphidz;       /* ∂φ_μ/∂z, μ=0,1,2,3 */
...
void main()
{
  /* fetch 23 neighboring values for cubics */
  getStencilFromVolumeTexture(volumeTexture, position);
  calcSplineCoeffsFromStencil();

  /* blossoming */
  mat4 blIn1  = deCasteljauLevel1(bcIn);    /* b^[1]_ijkl(φ) */
  mat4 blOut1 = deCasteljauLevel1(bcOut);   /* b^[1]_ijkl(φ̄) */

  vec4 blIn2  = deCasteljauLevel2(blIn1, bcIn);   /* b^[2]_ijkl(φ) */
  vec4 blOut2 = deCasteljauLevel2(blOut1,bcOut); /* b^[2]_ijkl(φ̄) */

  float b30 = dot(blIn2, bcIn);
  float b21 = dot(blIn2, bcOut);
  float b12 = dot(blOut2, bcIn);
  float b03 = dot(blOut2, bcOut);

  /* Newton root finding */
  vec3 roots = findRoots(b30,b21,b12,b03,isovalue);
  vec4 bcHit; /* barycentrics of intersection point */

  float tHit=-1;
  for each t in roots
  {
    /* barycentrics φ(x(t)) are found by linear interpolation */
    bcHit = mix(bcIn,bcOut,t);
    if (insideTetrahedron(bcHit))
    {
      tHit = t;
      break;
    }

  }
  if (tHit<0) return;  /* no hit found; discard fragment */
```

```
  /* derivative at intersection point */
  mat4 blHit1 = mix(blIn1,blOut1, tHit);              /* b^[1]_ijkℓ(φ(x(t))) */
  vec4 blHit2 = deCasteljauLevel2(blHit1, bcHit);     /* b^[2]_ijkℓ(φ(x(t))) */

  /* gradient computation */
  float nx = 3*dot(blHit2,dphidx);
  float ny = 3*dot(blHit2,dphidy);
  float nz = 3*dot(blHit2,dphidz);
  vec3 N = normalize(vec3(nx,ny,nz));

  ... /* shading of fragment, etc. */
}
```

Listing 6.1: Fragment shader for cell projection with cubic splines and on-the fly computation of B-coefficients.

6.1.3 Performance Analysis

In Figure 6.10 we plot millions of tetrahedra against frames per second for our variants of cell projection / ray casting algorithms based on trivariate splines. The numbers correspond to a worst case scenario where the surface is filling the entire screen and all tetrahedra are within the view port. Frame rates significantly increase for smaller view ports as well as for close inspection of the surface, since in the latter case, not all tetrahedra need to be processed. The largest isosurface in these tests consists of approximately 7.5 million tetrahedra within 650 000 relevant cubes. Since we use a 4 byte index to encode the position for each T in $\mathbf{T}_{\text{active},i}$ we need about 29 MByte of GPU memory for the geometry encoding in this case. For the variants using precomputed determining sets we further need to store 20 float values per relevant cube for quadratic super splines, which corresponds to circa 50 MByte in the above example, and for the cubic C^1-spline we need 56 float values per relevant cube, i.e., circa 139 MByte. If the spline coefficients are computed on-the-fly then the data set itself needs to be stored on the GPU.

As a rough estimate of shader performance we report that NVidia's tool *ShaderPerf* gives us a throughput of 143 million fragments per second for the most complex shader (cubic splines with on-the-fly coefficients) and 778 million fragments per second for the simplest shader (quadratic splines and precomputed determining sets). In our tests we achieve a peak performance of about 30 million Bézier tetrahedra per second for the variants using precomputed determining sets. In comparison, in [SWBG06] the authors achieve a rendering performance of about 11 million quadratic surfaces on similar hardware, but restrict themselves to only spheres, ellipsoids, and cylinders. In our own tests based on the approach in [LB06] we were able to render about 500 000 quadratic Bézier tetrahedra per second, where the main problem is that the screen space projections of the tetrahedra are done on the CPU. In addition, both approaches do not consider splines, but the much simpler case of unconnected polynomials. The GPU approach based on trivariate splines given in [SGS06] has a peak performance of only 850 000 quadratic Bézier tetrahedra. Further, since the authors ignore the spline struc-

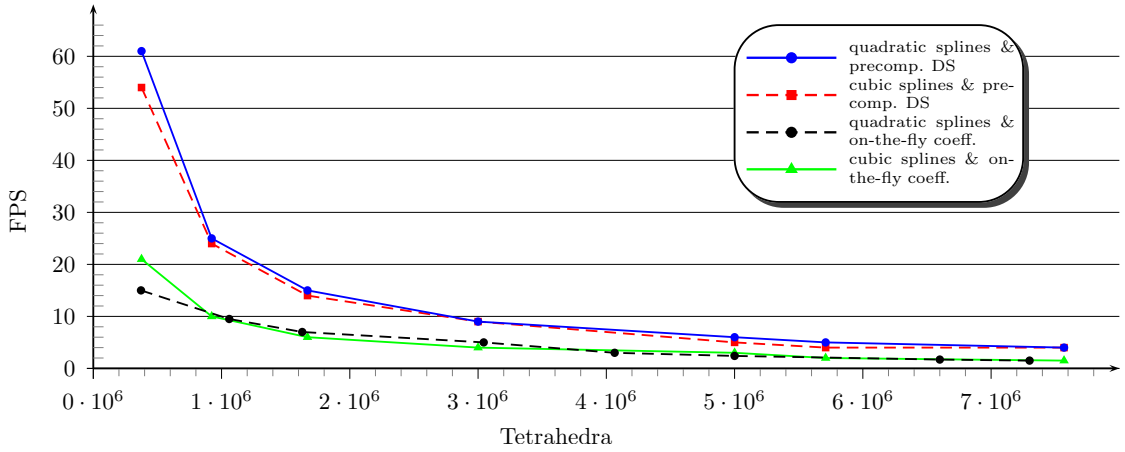

Figure 6.10: A plot of millions of tetrahedra against frames per second for four variants of trivariate splines rendering methods (quadratic splines and cubic C^1-splines with precomputed determining sets and with on-the-fly computation of coefficients). All timings were done on a NVIDIA GTX285 GPU and a 1000×1000 view port.

ture and encode the geometry in a naive way, this approach suffers from severe memory limitations.

The peak rate of 30 million tetrahedra corresponds to about 60 million triangles (the back facing triangles are culled in the triangle setup stage). This can be compared with the peak rate of modern GPUs which are capable of processing up to 300 million triangles in optimal conditions. Here, *optimal* refers to the size of the screen space projections of triangles as well as the overdraw, i.e., the number of fragments that fall onto one single pixel. In the rasterizer, the GPU processes batches usually consisting of 2×2 pixels in parallel. If the screen-space projection of a triangle is below this value then threads on the GPU diverge with the effect of a dropping performance. High overdraw affects performance since many computations are done for fragments which later get overwritten by fragments closer to the viewer. The raster operations unit (ROP) is the GPU stage that finally writes a pixel to the frame buffer. High overdraw is also a problem in the ROP since the number of units is limited (32 for a GTX 280 GPU) and frame buffer writes need to be serialized. The frame rates about linearly depend on the number of tetrahedra contributing to the surface indicating that vertex processing is one of the main performance bottlenecks. Consider an isosurface with five million tetrahedra, where for each tetrahedron six vertices need to be processed. On a 1000×1000 view port, this corresponds to 30 vertices per pixel. The performance of our shaders using precomputed determining sets is limited by these factors: a huge amount of small triangles and high overdraw. This is also reflected in Figure 6.10 since the red and blue curves almost coincide even though cubic splines have a shader complexity which is about twice as high as for quadratics.

We have lower performance using on-the-fly computation of spline coefficients, since in this case, more arithmetic is needed to obtain the coefficients from the data stencil.

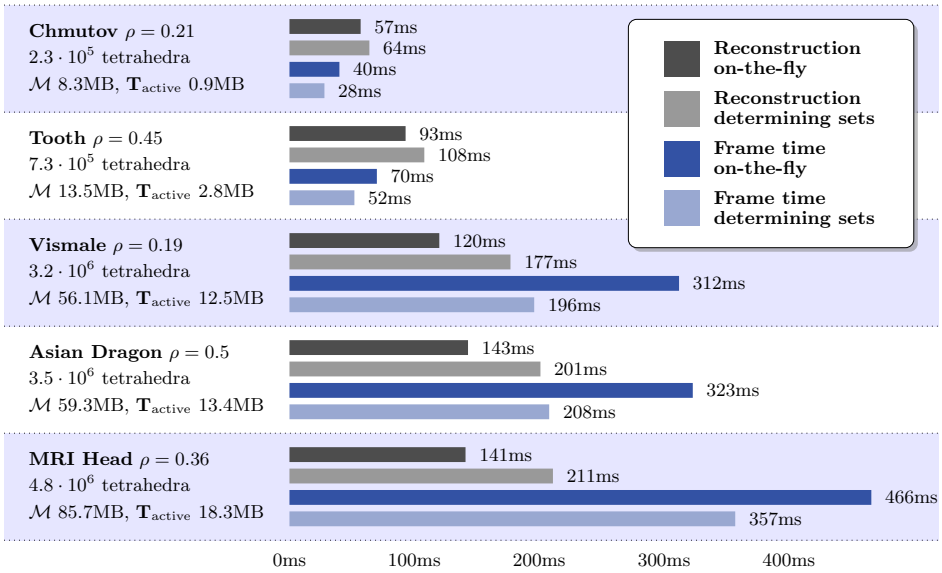

Figure 6.11: Reconstruction and frame times of cubic C^1-splines for a selection of data sets (see Figures 6.20, 1.1, 6.16). For each data set, the isovalue ρ, the number of active tetrahedra, the size of the determining set $\mathcal{M}$, and the size of $\mathbf{T}_{\text{active}}$, are given.

We furthermore need more texture fetches, which is 27 (instead of 5 for precomputed determining sets) in the case of quadratic super splines, 28 for quadratic C^1-splines on truncated octahedral partitions, and 23 texture fetches for cubic C^1-splines (14 for precomputed determining sets). Again, the frame rates of quadratic and cubic splines are very similar (the black and green curves in Figure 6.10), indicating that the higher arithmetic complexity of cubics is almost compensated for by fewer texture fetches.

By using instancing we need $|\mathbf{T}_{\text{active}}|$ 4-byte words to implicitly encode the geometry of the tetrahedra on the surface. The memory requirements for the precomputed determining sets are determined by the number of active cubes $|\mathbf{Q}_{\text{active}}|$. Since on average 12 tetrahedra are relevant on each $Q \in \mathbf{Q}_{\text{active}}$ the memory requirements are then given by $|\mathbf{T}_{\text{active}}| \cdot |\mathcal{M}_Q|/12$, where $|M_Q|$ is 20 for quadratic super splines, and 56 for cubic C^1-splines, respectively. Storing the precomputed determining set needs less memory than the complete data set of n^3 values if $|\mathbf{T}_{\text{active}}| < n^3 \cdot 12/|\mathcal{M}_Q|$. For example, for a data set of size 256^3 and quadratic super splines this corresponds to about 10 million tetrahedra, and for cubic C^1-splines we have 3.59 million tetrahedra. Above these values it is more memory efficient to compute the spline coefficients on-the-fly from the volume data.

The table in Figure 6.11 exemplarily summarizes the performance of cubic C^1-splines for a selection of data sets and lists the isovalues ρ, the number of active tetrahedra, as well as the size of the determining set $\mathcal{M}$ and the geometry encoding $\mathbf{T}_{\text{active}}$. Timings were recorded on a GeForce GTX 285 (240 unified shader units) and CUDA 2.2. For each data set, the first two bars show the timings of our GPU kernels (see Section 6.1.1), which are invoked when the surface needs to be reconstructed. The on-the-fly computation

of coefficients is slightly faster than preparing the determining sets, since less data is written. The most expensive part in the reconstruction is the classification, i.e., the determination of the array $\mathbf{Q}_{\text{class}}$. This could be improved by using appropriate spatial data structures, e.g., min-max octrees, but the recursive nature of these data structures makes an efficient GPU implementation challenging. In addition, the data structures have to be rebuilt if the data itself changes over time, which is not necessary in our simpler approach. Still, for our largest data sets the reconstruction times are in a range of a few hundred ms and in most cases even below the rendering times of a single frame. Compared to optimized CPU reconstruction based on octrees we achieve significant speed-ups of up to two orders of magnitude. The per-frame rendering times are also given for a 1000×1000 view port with the surface filling the entire screen.

6.2 An Image Based Volume Ray Casting Approach

The performance analysis of our hybrid cell projection / raycasting algorithm and the recent development allowing for GPU programs with an unlimited number of instructions as well as bit logical and shift operations introduced with Shader Model 4.0 has led to a pure *geometry free* volume ray casting approach for trivariate splines. For simplicity, we only consider splines on the type-6 partition. Similar to standard GPU volume ray casting, e.g., [KW03], we rasterize the screen-space projection of the volume's bounding box. For each pixel we use the fragment shader to traverse a ray throughout the volume in a large loop. Since our trivariate splines are based on more complex tetrahedral partitions instead of simple cubic partitions in the case of tensor-products special care has to be taken in order to make optimal use of the GPU resources and to achieve interactive or real-time frame rates. An efficient empty space leaping technique is of major importance to avoid costly calculations for empty cells, see Section 6.2.1. Further, we need to quickly traverse a ray through a relevant cube Q, i.e., we have to quickly determine the ray entry and exit points of a ray w.r.t. the tetrahedra within Q, see Section 6.2.2. Moreover, it is of crucial importance to keep the local register usage low and to avoid dynamic array indexing. Otherwise, current GPUs fall back on using slow and non-cached global memory which has an immediate negative impact on the performance.

6.2.1 Empty Space Leaping

Fast ray traversal for cubic grids using a 3D extension of the DDA algorithm for line rasterization has been given by Amanatides and Woo [AW87]. In order to significantly reduce the overhead for non-contributing cells we use an empty space leaping technique based on octrees, where the rays are subdivided into non-uniform sections with each section starting at a voxel face. Traversal schemes of this kind have been discussed by Cohen and Sheffer [CS94]. This basic principle has been improved in the anisotropic *chessboard distance* voxel traversal by Šrámek and Kaufman [vK00], but at the cost of a more complex preprocessing. GPU implementations of these algorithms where dynamic branching is minimized are discussed in [EI07].

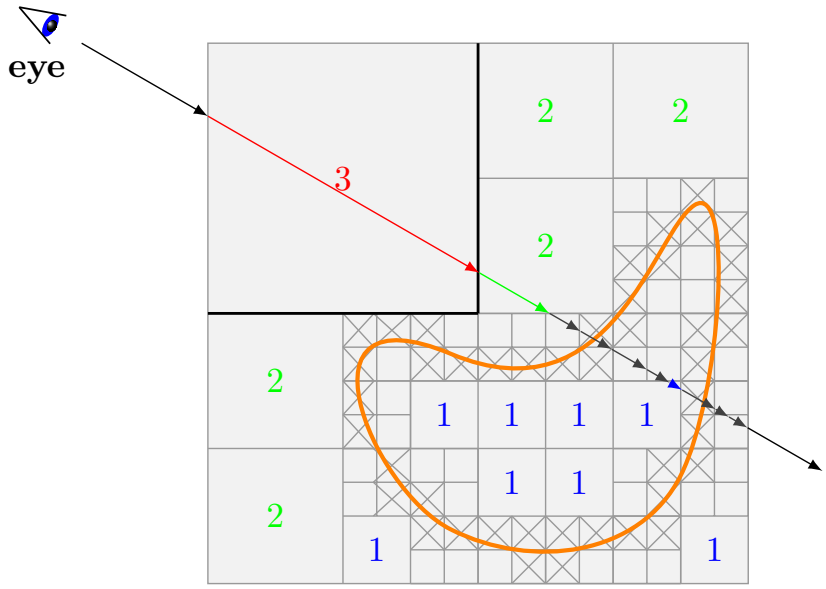

Figure 6.12: 2D illustration of fast empty space leaping based on octrees. For each empty macro region we store the corresponding skip index (colored numbers). The lines characterized by the Heaviside functions are shown in black. Ray sections are shown as colored arrows. Only the non-empty cells on the finest level are tested for intersections with the isosurface (orange).

For each change of isovalue ρ we have to rebuild a *skip volume* data structure which encodes the sizes of the empty octree cells (*macro regions*) as well as the relevant tetrahedra for non-empty cells. To do this, we traverse a min-max octree starting from the root and for each node we test if $\rho \notin [f_{\min}, f_{\max}]$. In this case, we found an empty (macro) cube and store a 0 on the finest level, a 1 on the next level, etc. Using this *skip index* $\imath$ the side length of a (macro) cube Q relative to the size of the cells at the leaves is given by $|Q| = 2^{\imath}$, see Figure 6.12. Otherwise, we recursively traverse the next level of the node. Note that each macro cube $Q := Q_{ijk}$ covers the $|Q|^3$ leave nodes with indices varying from $i, \ldots, i + |Q| - 1$, $j, \ldots, j + |Q| - 1$, and $k, \ldots, k + |Q| - 1$. On the finest level we further determine the relevant tetrahedra within each active cube Q. For splines on the type-6 partition we can encode the relevant tetrahedra with a 24 bit *tet code* where the rth bit is set if tetrahedron $T_r \in Q$, r in $0, \ldots, 23$, is active. We use the finest level of the octree, i.e., we store the skip indices as 8 bit numbers at the leaves of a full octree with the same dimension as the volume data set and combine them with the 24 bit tet codes in one texture.

For the traversal of the macro voxels we first notice that we can reduce the number of candidate planes which have to be tested for intersections from six to three. In each of the three principal directions the ray segment leaves the voxel at the plane characterized by the *Heaviside* function

$$H(x) = \begin{cases} 0, & x < 0, \\ 1, & x \geq 0. \end{cases}$$

Let $\mathbf{d} = (d_x, d_y, d_z)^\mathsf{T}$ be the ray direction, then we have to consider the left plane of the voxel if $H(d_x) = 0$, and the right plane for $H(d_x) = 1$, analogously for d_y and d_z, see Figure 6.12. For each (macro) cube $Q := Q_{ijk}$ with side length $|Q| = 2^i$ we thus obtain the ray parameters of the intersections with the axis parallel planes of Q by

$$t_x = \frac{i + |Q| \cdot H(d_x) - o_x}{d_x},\ t_y = \frac{j + |Q| \cdot H(d_y) - o_y}{d_y},\ t_z = \frac{k + |Q| \cdot H(d_z) - o_z}{d_z},$$

where $\mathbf{o} = (o_x, o_y, o_z)^\mathsf{T}$ is the ray origin. We take the smallest $t \in \{t_x, t_y, t_z\}$ and proceed to the next grid cell $\widetilde{Q}$ where the cell indices on the finest octree level are found by

$$i = \lfloor o_x + t \cdot d_x + \epsilon \cdot \mathrm{sign}(d_x) \rfloor,$$

analogously for j and k, with a small offset $\epsilon > 0$ added for numerical reasons, see [EI07]. If the tet index equals zero we have to continue the traversal of empty (macro) cubes and the corresponding macro cube index is given by the next decimal multiples $\lfloor i/|\widetilde{Q}| \rfloor \cdot |\widetilde{Q}|$, analogously for j and k. Otherwise, we have found a relevant cube and proceed by traversing the tetrahedra of $\widetilde{Q}$.

6.2.2 Tetrahedra Traversal

We can quickly determine the exit point $\tilde{\mathbf{v}}$ of the viewing ray w.r.t. a generic tetrahedron T using the four plane equations of T. Let $\mathbf{v} = (v_x, v_y, v_z, 1)^\mathsf{T} \in \mathbb{R}^4$ be the entry point of the ray. Note that the coefficients of the plane equations correspond to the rows of the precomputed matrix M_T^{-1}, see Section 3.3.1. We can thus consider the components of $\boldsymbol{\phi}(\mathbf{v})$ as the results of plugging $\mathbf{v}$ into the four plane equations of T and compute the barycentric coordinates as $\boldsymbol{\phi}(\mathbf{v}) = M_T^{-1}\mathbf{v}$. Analogously, we compute the directional coordinates of the ray direction $\mathbf{d} = (d_x, d_y, d_z, 0)^\mathsf{T} \in \mathbb{R}^4$ as $a(\mathbf{d}) = M_T^{-1}\mathbf{d}$ which corresponds to plugging the direction vector into the four plane equations. The intersection of a point on the ray with the μth plane of T, $\mu = 0, 1, 2, 3$, is now characterized by

$$\boldsymbol{\phi}(\mathbf{v}) + \alpha_\mu \cdot a_\mu(\mathbf{d}) = 0, \quad \alpha_\mu \in \mathbb{R}.$$

Solving for α_μ,

$$\alpha_\mu = -\frac{\boldsymbol{\phi}(\mathbf{v})}{a_\mu(\mathbf{d})}.$$

We now seek for the smallest $\alpha := \min(\alpha_\mu, \alpha_\mu > 0)$ and compute the barycentric coordinates of the exit point $\tilde{\mathbf{v}}$ as

$$\boldsymbol{\phi}(\tilde{\mathbf{v}}) = \boldsymbol{\phi}(\mathbf{v}) + \alpha \cdot a(\mathbf{d}).$$

If T is not relevant, i.e., the respective bit in the tet code is not set, we proceed with the next tetrahedron. The neighboring tetrahedron of T is found from a constant table addressed with T's index and the plane index μ. Otherwise, we have to determine the B-coefficients of $p = s|_T$. This can be done by either repeated averaging of the appropriate coefficients from the determining set, or we directly compute the coefficients

from the stencil. For intersection with the polynomial p we proceed using blossoming as described in Section 6.1.2 with the exception that clipping of intersection points with T is simplified: by using the exit point instead of an arbitrary point on the ray we just have to check if $0 \leq t \leq 1$.

The pseudo-code of our fragment program is shown in Listing 6.2.

```
uniform usampler3D skipVolume;
uniform sampler3D  volumeTexture;

in vec3 rayOrigin; /* point on the bounding box of the data set */
in vec3 rayDir;

void main()
{
  rayDir = normalize(rayDir);
  ivec3 cubeIndex = ivec3(floor(rayOrigin));

  /* initialize octree traversal */
  vec3 rayDirInv = 1.0 / rayDir;
  vec3 signR = sign(rayDir);
  vec3 H = clamp(signR,0,1); /* Heaviside */

  while (inside volume)
  {

    unsigned int skipCode = texelFetch(skipVolume, cubeIndex);
    unsigned int tetCode = skipCode >> 8;

    if (tetCode) /* we have relevant tetrahedra in Q */
    {

      getStencilFromVolumeTexture(volumeTexture, cubeIndex);
      calcDeterminingSetFromStencil();
      /* translate Q to the origin */
      vec4 vIn = vec4(rayOrigin - (cubeIndex + vec3(0.5)),1);
      /* get index of first T along the ray */
      int tetIndex = getTetIndexFromPoint(vIn);

      while (inside Q)
      {
        mat4 Mi = getInverseTetrahedronMatrix(tetIndex);
        vec4 bcIn = Mi*vIn;

        /* find exit point */
        vec4 a = Mi*rayDir;
        vec4 alpha = -bcIn/a;
        float alphaMin = findMinimumPositiveAlpha(alpha);

        if (tetCode & (1 << tetIndex))) /* T is relevant */
        {
          calcRemainingCoeffsFromDeterminingSet(tetIndex);
          vec4 bcOut = bcIn + alphaMin*a;
          if (intersectionFound(tetIndex))
          {
            shadeFragment();
            return;
          }
        }
        /* find next tetrahedron */
        int planeIndex = findPlaneIndex(alphaMin);
        tetIndex = getNeighboringTetrahedron(tetIndex, planeIndex);
```

```
        if (tetIndex == -1)
          break;          /* we have left Q */

        vIn = vIn + alphaMin*rayDir; /* advance tet entry point */

      } /* end while (inside Q) */
    } /* end if (tetCode) */

    /* empty space leaping */
    /* directly go to neighboring "macro" cube */
    int skipIndex = skipCode & 0xFFFF;
    int cubeSize = 1 << skipIndex;            /* |Q| */
    ivec3 macroCubeIndex = (cubeIndex >> skipIndex.xxx) << skipIndex.xxx;
    vec3 t = (macroCubeIndex + H * cubeSize - position) * rayDirInv;
    float tMin = min(t.x,min(t.y,t.z));
    /* advance ray */
    cubeIndex = ivec3(floor(rayOrigin + tMin * rayDir + eps * signR));
  } /* end while (inside volume) */
}
```

Listing 6.2: Fragment shader for ray traversal and quadratic super splines

One problem of this approach are diverging GPU threads, e.g., a ray in a batch hits relevant cubes while the others traverse empty space. In this case, the threads have to wait for the slowest ray to finish and GPU utilization is far below the optimum. We thus further improved the performance of our volume ray casting for trivariate splines about more than 50% by using two rendering passes. In the first pass we determine the ray parameter t_{start} of the intersection with the first relevant cube, and we analogously determine t_{end}, which corresponds to the exit point of the last relevant cube along the ray. In the second pass, ray traversal for each fragment directly starts with the first relevant cube using t_{start}, and fragments with no relevant cubes along the ray do not need to be further processed. This way, the number of diverging threads in a batch is significantly reduced and GPU utilization is improved.

The two rendering passes do not completely eliminate diverging threads. Still, rays exists that take more complex paths than others, in particular rays that hit a relevant cube Q along their way but do not have an intersection with the isosurface within Q, see Figure 6.13. To alleviate this problem we ported our kernel to CUDA where we can make use of a *persistent thread* pool, see also the recent paper by Aila and Laine [AL09]. Here, each kernel thread just runs in an infinite loop. If ray traversal for one pixel is finished the thread does not wait for the others but immediately continues by restarting ray traversal with a new ray obtained from the thread pool. This way, GPU utilization is further improved resulting in a significant performance gain, see Section 6.2.3.

6.2.3 Performance Analysis

In Figure 6.14 we plot the frames per second against millions of tetrahedra on a 1000×1000 view port. The blue curve is replotted from Figure 6.10 and extended for comparison with the hybrid cell projection / ray casting approach. The red dashed curve shows the performance of our image based algorithm using shader programs in the OpenGL graphics pipeline. While the blue curve declines with increasing number of tetrahedra the frame rates of our image based approach are much less dependent on the size of the

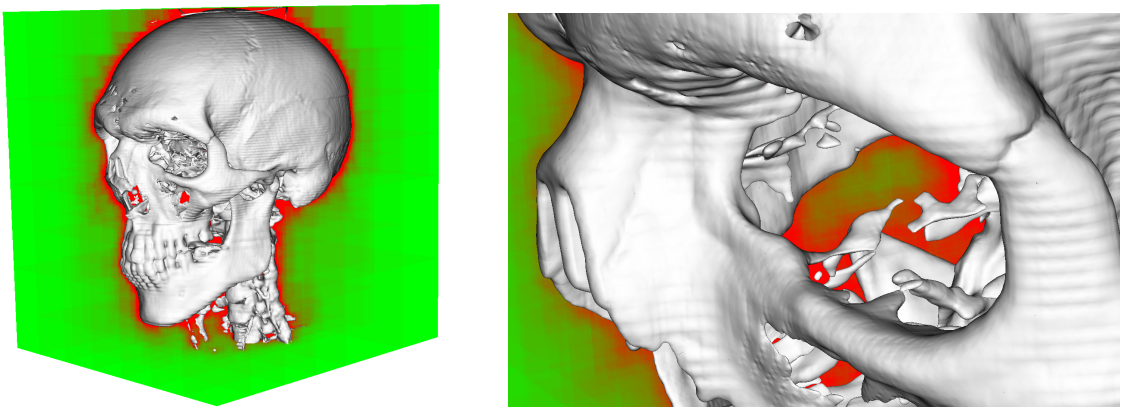

Figure 6.13: Illustration of ray path complexity. Green denotes regions where rays do not hit any relevant cubes. These rays are already eliminated in the first rendering pass. Red denotes regions where rays traverse relevant cubes but miss the isosurface. A more saturated red tone indicates more relevant cubes along the ray path. The right picture is a close-up into the eye of the skull.

isosurface and are roughly three to four times higher for large surfaces consisting of several millions of tetrahedra. For very large isosurfaces (more than 12 million tetrahedra) performance of the hybrid approach drops below one frame per second while the frame rates in the pure image based ray casting stay almost constant.

The performance gain of our CUDA kernels using persistent threads (green curve in Figure 6.14) is even more impressive. Here, we achieve real-time frame rates (more than 25 FPS) even for the largest isosurfaces under consideration with more than 20 million tetrahedra and performance is roughly three times higher than using OpenGL even though the kernels run on the same hardware.

The complex CUDA architecture is organized as a number of multiprocessors (MP) each consisting of eight scalar processor cores. In order to hide memory access latencies each MP is capable of running hundreds of concurrent lightweight threads with thread scheduling being done by the hardware with no overhead. This concept is labeled SIMT (single instruction, multiple threads) in analogy to SIMD architectures. On each MP the cores further share a certain amount of registers and on-chip shared memory. The DRAM on the graphics board is structured into non-cached local and global memory, and cached constant and texture memory. Local memory is only accessible by the multiprocessors and its use is determined by the CUDA compiler if the register space is insufficient for the code to run or when dynamic array indexing is used. For maximum performance the use of this off-chip and non-cached local memory should be avoided. Further aspects that affect performance on a CUDA device are related to memory access patterns and latencies arising from register read-after-write dependencies. For a more in-depth discussion of the CUDA architecture and performance guidelines see [NVI09a, NVI09b].

We use texture memory for storing the volume data set and the acceleration data structure. Constant tables such as the plane equations of the generic tetrahedra are

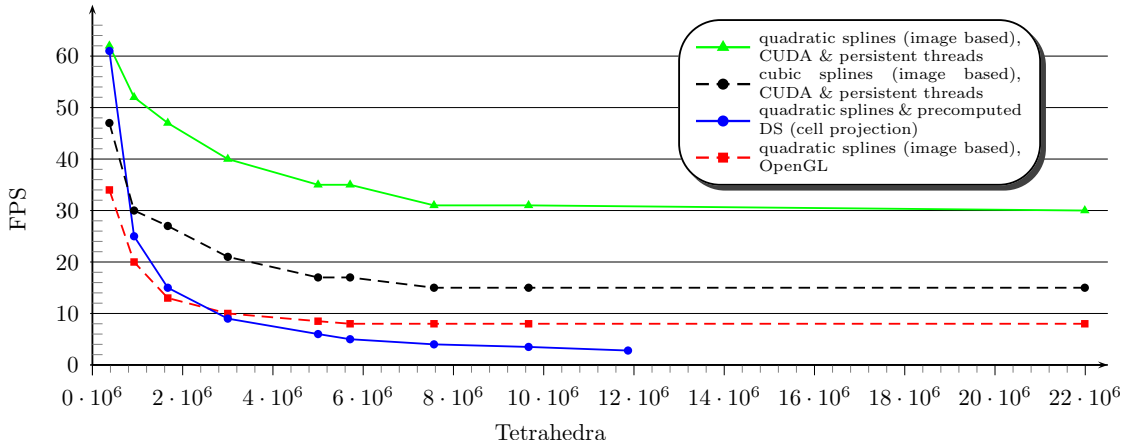

Figure 6.14: A plot of millions of tetrahedra against frames per second for several isosurface visualization methods based on trivariate splines. *Green curve*: image based ray casting for quadratic super splines using CUDA and persistent threads. *Black curve*: image based ray casting for cubic C^1-splines using CUDA and persistent threads. *Red curve*: image based ray casting for quadratic super splines using OpenGL. *Blue curve*: hybrid cell projection / ray casting for quadratic splines using precomputed determining sets (replotted from Figure 6.10). All timings were done on a NVIDIA GTX285 GPU and a 1000×1000 view port.

stored in constant memory. The limited number of registers is one of the main factors which affect the performance of our CUDA kernels. To reduce the number of registers for each thread we use on-chip shared memory to store the coefficients of the determining set. Then, our kernels for quadratic super splines use 70 registers. The thread scheduler can optimize register memory bank conflicts when the number of threads running on one MP is a multiple of 64. On the NVidia GTX 280 with 16384 registers per multiprocessor the optimal number of threads that should be started on each MP is thus 192. For cubics our kernels need 108 registers corresponding to 128 concurrent threads on each MP. We further expect our kernels to scale well with increasing hardware resources. The next generation GPUs by NVidia double the number of scalar processor cores from 240 (GTX 200 series) to 480 (GTX 400) which should be reflected in a significant performance gain.

For the sake of simplicity of the preprocessing and to achieve a constant number of texture accesses in the traversal of macro-cubes we store the skip codes at the leaves of a full octree. The memory requirements of our acceleration structure are thus equal to the size of the volume data set. On GPUs with 1 GByte texture memory we can therefore handle data sets with up to 512^3 voxels.

6.3 Results

We develop the first GPU-based algorithms for high quality rendering from volume data using trivariate splines, where we exploit the spline structure and uniformity of the underlying tetrahedral partitions in order to achieve high frame rates for real-world

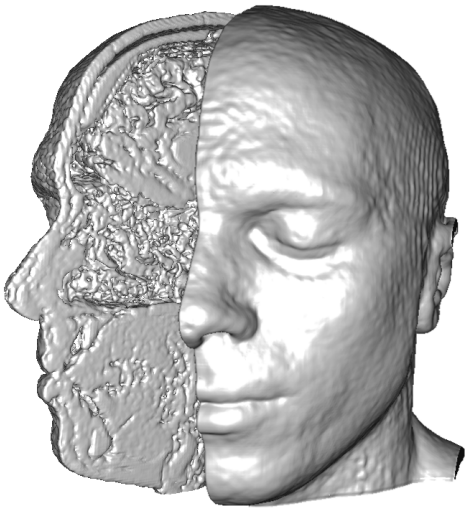

Figure 6.15: MRI scan ($192^2 \times 126$ voxels, approximately 4.8 million tetrahedra) volume clipped with the plane $x = 0$.

data sets. In particular, we give the first visualizations of quadratic and cubic quasi-interpolating C^1-splines. We have shown that interactive and high-quality visualizations of volume data with varying isosurfaces based on trivariate splines can be efficiently performed on modern GPUs. Our approaches benefit strongly from the mathematical properties of the splines such as small data stencils and stable evaluation of the low-degree B-form polynomials. We have given an efficient algorithm based on a hybrid cell projection / ray casting as well as a pure image based ray casting. Our methods scale well with the fast developing performance of modern graphics processors and will directly benefit from increased numbers of multiprocessors and texture units. The proposed algorithms can be used for an interactive variation of isolevels as well as for applications where the data itself varies over time, e.g., simulations and animations. The visual quality and performance of our approaches are more than competitive with existing techniques for isosurface visualization from volume data.

We demonstrate our results with a series of data sets. *VisMale*, *Tooth*, and *Foot* (see Figure 1.1), and *Aneurismn* (see Figure 2.2, top) are medical data sets publicly available from the US Library of Medicine, and the Universities of Tübingen and Erlangen, respectively. The *Bonsai* (see Figure 2.2, bottom) and *Neghip* (Figure 6.18) data sets are also available from the University of Tübingen. Figure 6.15 shows a MRI scan of a head clipped with the plane $x = 0$. The Asian Dragon in Figure 6.16 is generated from a signed distance function on the original triangle mesh. Figures 6.17, 6.19, 6.20, and 6.21 are examples of synthetic data obtained from sparsely sampled smooth functions. In Figure 6.22 we give some examples of surfaces reconstructed from Cryo-EM data. All results demonstrate the high visual quality and smooth shading of our method.

Standard techniques from Computer Graphics are straightforward to apply to our spline renderings, see for example Figures 1.1, 6.16, and 6.19, where we illustrate trans-

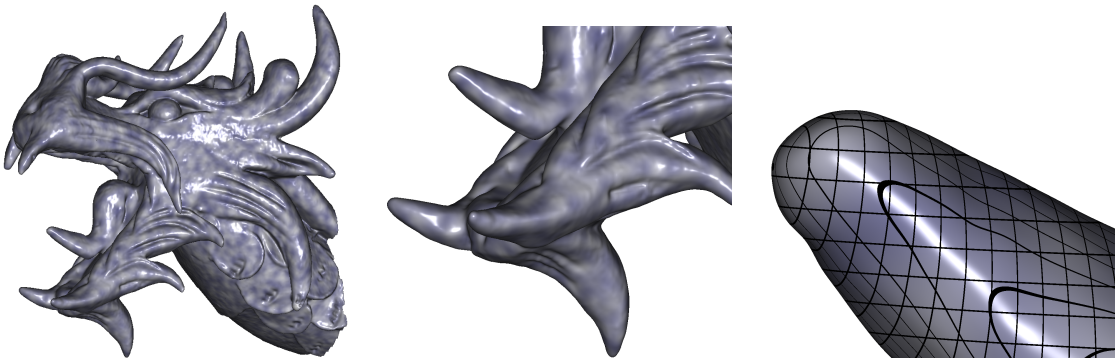

Figure 6.16: Ray casted isosurface of the Asian Dragon head (256^3 voxels, approximately 3.5 million tetrahedra) with noise-based procedural texturing. Right : close-up into the Dragons' mouth where the C^1-continuous boundary curves on the outside of each cube Q in $\square$ are shown in black

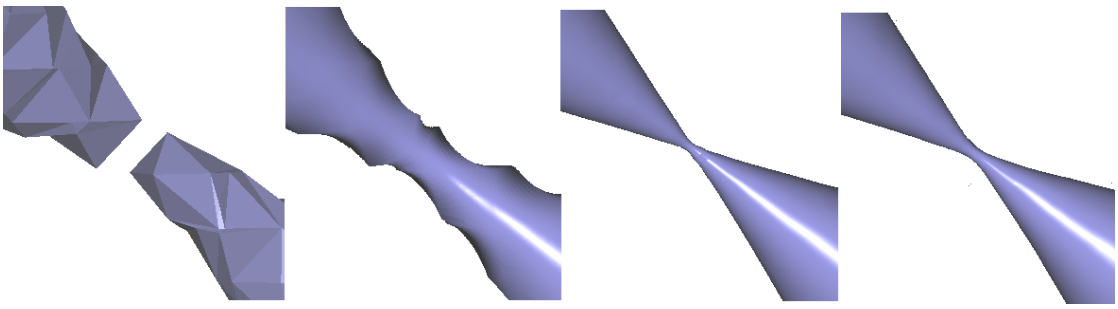

Figure 6.17: Visual comparison of isosurfaces demonstrated with a close-up on one of the spikes of Barth's sextic function (see Figure 5.7). *From left to right*: Marching Cubes, trilinear ray casting, cubic C^1-splines on type-6 partitions, quadratic C^1-splines on truncated octahedral partitions.

parency, texturing, and reflection mapping, respectively. For the cell projection approach we can obtain transparency at the cost of additional rendering passes, a technique which is known as *depth peeling* [Eve01, LHLW09]. With the pure ray casting rendering of transparent surfaces is very simple. Moreover, our splines have an inherent level of detail, i.e., we can zoom far in without the need to refine the spline model, and visualizations remain smooth (see Figures 2.2, 6.16, right, and 6.17). The low total degree of the piecewise polynomials allows us to obtain precise intersections even for the objects' silhouettes without resorting to interval refinements or similar approaches. Precise intersections are also needed for procedural texturing (see Figures 6.16, 6.19, 6.20, and 6.21), and for volume clipping with arbitrary planes and surfaces (see Figures 6.15 and 6.23). Furthermore, the obtained intersections are exact w.r.t. z-buffer resolution which allows us to combine ray casted isosurfaces with standard object representations, e.g., triangle meshes.

We proceed with a visual comparison of our results with standard approaches. The surfaces in Figure 2.2, left, are reconstructed with trilinear ray casting. The same isolevel obtained from cubic C^1-splines is shown on the right. Figure 6.17 shows reconstructions

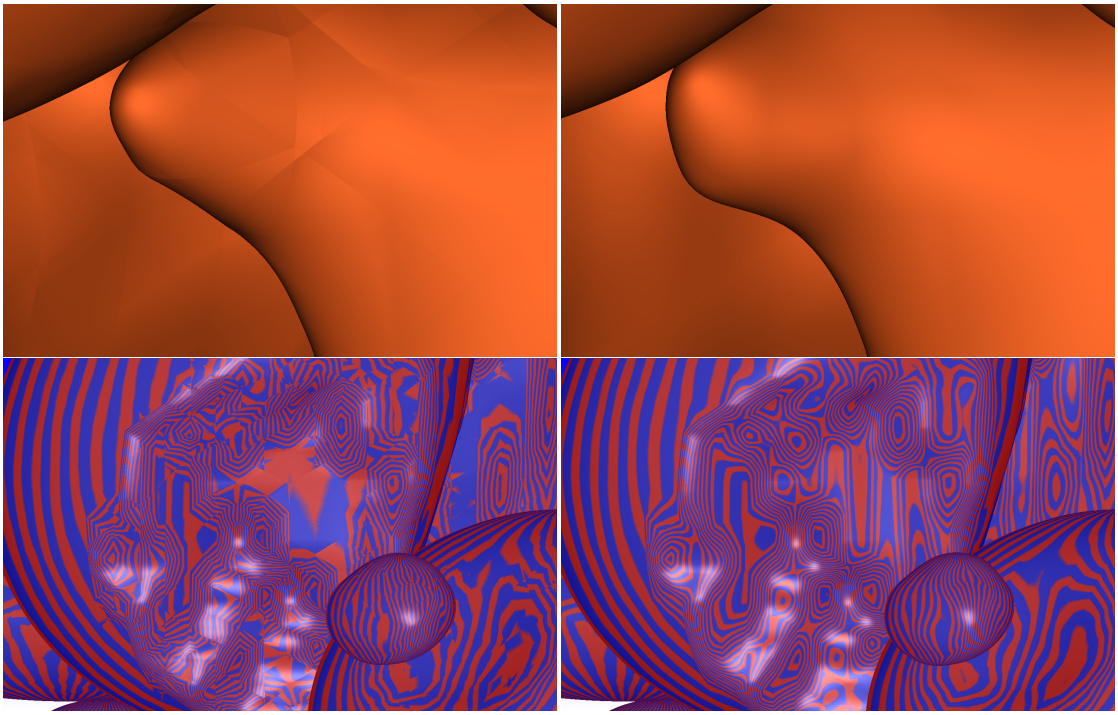

Figure 6.18: Visual comparisons of quadratic super splines (*left*) and cubic C^1-splines (*right*). *Top*: close-up into the *aneurism* data set (256^3 voxels). *Bottom*: the reflection lines on the *neghip* data set (64^3 voxels) in a highly oscillating region.

obtained from Marching Cubes, trilinear ray casting, quadratic C^1-splines on truncated octahedral partitions, and cubic C^1-splines, respectively with a close-up into one of the spikes of the Barth sextic function. As can be seen from the figures trivariate splines appear smoother and much more natural having an overall higher visual quality with almost no tessellation artifacts. For instance, using our approach the leaves of the Bonsai in Figure 2.2 possess improved boundaries with almost perfect silhouettes while the trilinear ray casting fails to reproduce the fine structures. Because of their approximating nature our methods are quite insensitive to noise and automatically avoid undesired oscillation and stair casing. In Figure 6.18 we visually compare quadratic super splines with cubic C^1-splines on the type-6 partition. The close-up at the top shows that C^1-splines reduce the undesired visibility of the underlying tetrahedral partition in areas with high curvature. Moreover, Figure 6.18, bottom, illustrates that the reflection lines of cubic C^1-splines have an improved appearance.

Our cell projection outperforms previous GPU approaches for quadratic super splines, e.g., [SGS06], by almost two orders of magnitude and has significantly less memory requirements for spline and geometry encoding. In contrast to the cubic C^0-splines obtained from MLS approximation [KOR08], our quasi-interpolants allow for on-the-fly computation of the polynomial coefficients directly on the GPU. Therefore, we do not need to inflate the data prior to the visualization and can thus handle significantly larger

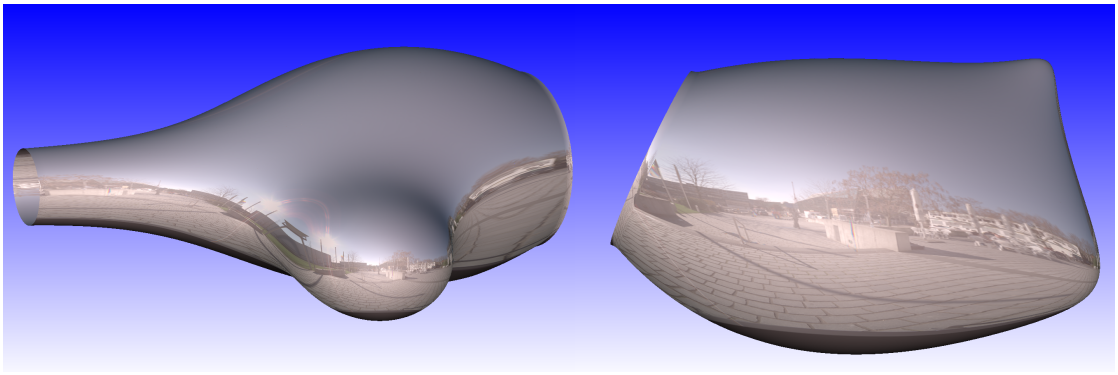

Figure 6.19: Examples of a reflection mapping applied to isosurfaces reconstructed using quadratic super splines. The pictures show different isolevels of an exponential function sampled on a 64^3 grid.

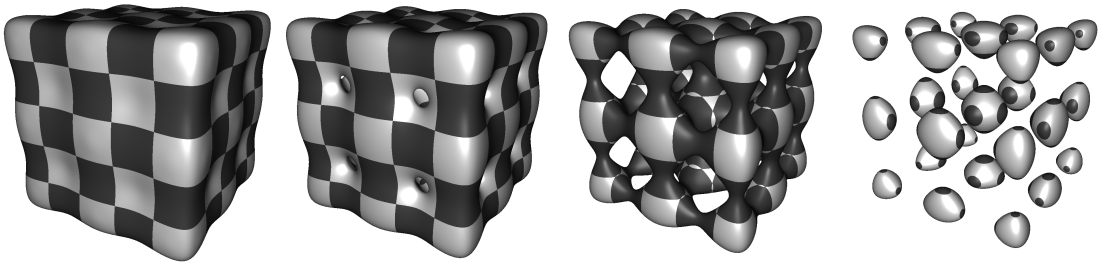

Figure 6.20: Varying isosurfaces of a synthetic function of *Chmutov* type, $f(x,y,z) = x^{16} + y^{16} + z^{16} - \cos(7x) - \cos(7y) - \cos(7z)$, sampled on a sparse grid (64^3 data points) with real-time reconstruction and rendering times.

data sets with current hardware. Since we use B-form polynomials and blossoming for fast intersection and derivative calculation our kernels are at least on par with current approaches, e.g., [SWBG06, LB06], for visualizations of unconnected quadratic and cubic surfaces which is a much simpler setting than trivariate splines. Further, to allow for an interactive change of isosurface for large data sets we use CUDA in the reconstruction step, i.e., the determination of the relevant tetrahedra. Both, memory requirements and processing speed are hereby comparable to GPU marching cubes for triangle mesh reconstruction, e.g., [TSD07]. We have also shown the principal limitations of a hybrid cell projection / ray casting approach which lies in the nature of current GPUs to process batches of 2×2 fragments. This becomes a problem if the screen space projections of triangles are below this value and GPU threads diverge. Since this is also a problem for the widespread refinement and subdivision methods such as PN triangles [VPBM01] or Loop subdivision [Loo87] it is very likely that this issue will be considered in next generation GPUs. Another problem is high overdraw, where many calculations are done for fragments that get overwritten and the ROP stage becomes an additional bottleneck. As an option we could sort the tetrahedra in each frame front-to-back to improve hardware-based *early z-culling* for invisible fragments [NVI08b]. Gradients at the inter-

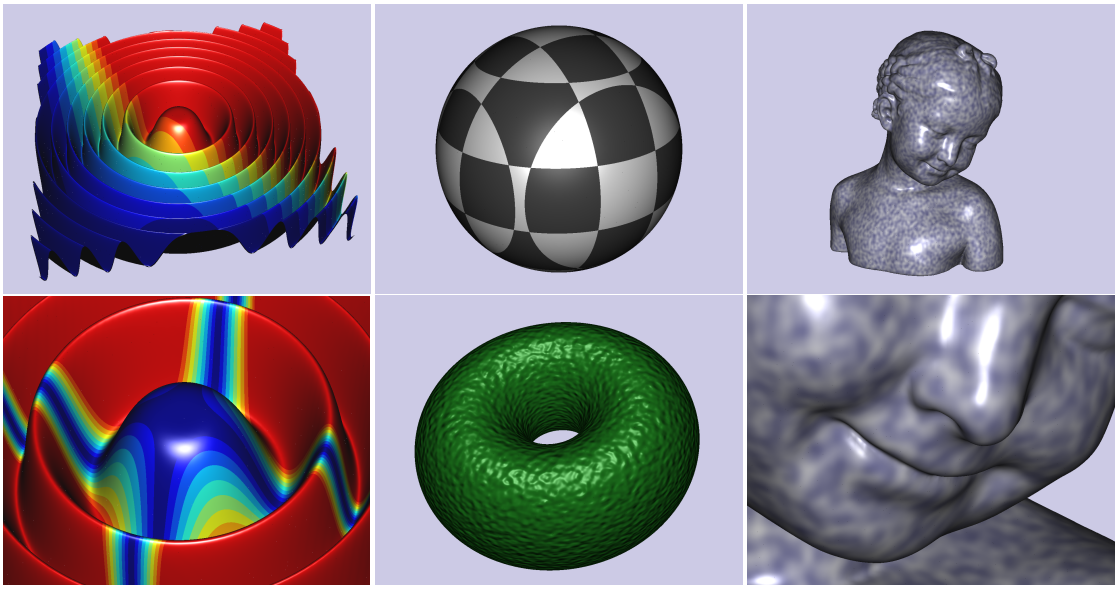

Figure 6.21: Examples of smooth renderings from synthetic data sets with different texturings.

sections can be stored in an intermediate buffer for a second *deferred shading* [Koo07] rendering pass to improve performance for, e.g., complex light/surface interaction, or ambient occlusion.

Our direct ray casting approach is better suited for very large isosurface consisting of several ten million tetrahedra. We make extensive use of an efficient empty space skipping structure, the uniformity of the tetrahedral partition, and new GPU hardware features such as bit logical and shift operations available since Shader Model 4.0. Direct ray casting does not suffer from the overdraw problem. Diverging threads are significantly reduced by a first rendering pass determining the first and the last relevant tetrahedron along the ray. We can further improve GPU usage by using a pool of persistent CUDA threads with each thread proceeding with a new ray once traversal of the current ray is finished. This results in roughly three times higher frame rates compared to the ray casting in the OpenGL graphics pipeline and 30 times higher frame rates than cell projection for very large surfaces. We achieve real-time frame rates for the largest isosurfaces under consideration on current graphics hardware. The performance and memory requirements are comparable to GPU ray casting using trilinear and higher order, e.g., tricubic, tensor products [HSS*05]. Frame rates are significantly better than reported on box splines [FEVM10] or volume ray casting based on MLS reconstruction [LGM*08]. Since our acceleration structure stores the skip codes at the leaves of a full octree we need twice as much GPU memory as for the data set itself. With limited GPU memory streaming techniques such as bricking [PSL*98] can be used. Memory

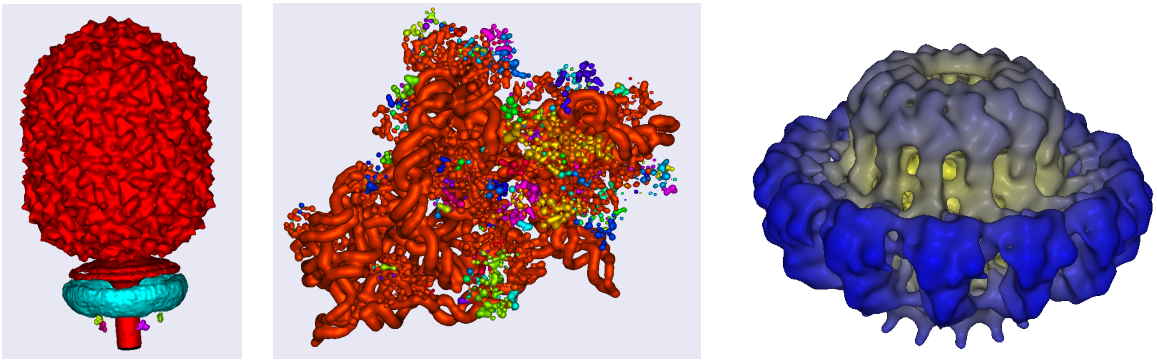

Figure 6.22: Examples of isosurfaces extracted from Cryo-EM data. *Left*: Bacteriophage Φ29 virus (171^3 voxels, 2 million tetrahedra). *Middle*: Bacterial 30S Ribosome subunit (128^3 voxels, 1 million tetrahedra). *Right*: Viral portal channel protein (100^3 voxels, 1.2 million tetrahedra). The virus and ribosome were reconstructed with quadratic super splines, and the channel protein with the quadratic C^1-spline on truncated octahedral partitions.

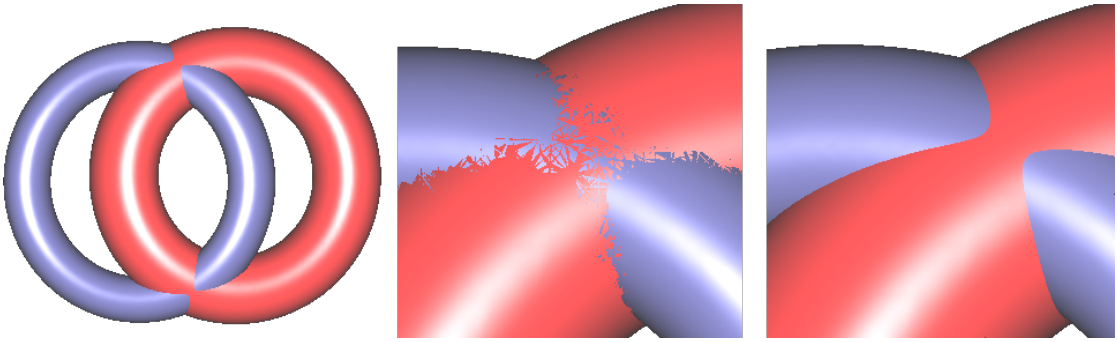

Figure 6.23: Intersecting isosurfaces reconstructed with the quadratic C^1-splines on truncated octahedral partitions. *Middle*: without depth adjustment. *Right*: intersections are exact if the correct depth values are used.

usage can be further improved by employing a sparse octree at the cost of non-constant memory accesses for macro cube traversal and higher overhead for stream compaction in the preprocessing. Note that simple trilinear ray casting needs more GPU memory than our approach if derivatives are precomputed to avoid central differencing.

Overall, cubic splines on the type-6 partition have roughly three times higher overhead than the quadratic super splines. Cell projection is straightforward to apply to splines on truncated octahedral partitions where approximately 50% more tetrahedra are relevant compared to splines on the type-6 partition. The complexity of the truncated octahedral partition makes a pure image based ray casting more challenging to implement for this type of splines.

We conclude this chapter with some further remarks on the implementation issues.

Remark 4. *The coefficients of the determining sets of the quasi-interpolating splines can be directly obtained from the volume data by texture fetches with hardware-accelerated*

linear interpolation. In the case of quadratic super splines this corresponds to only 20 texture accesses compared to the 27 texture fetches needed to obtain the values of the data stencil by nearest neighbor interpolation. Nevertheless, our experience with current GPUs shows that it is approximately 30% faster to fetch the 27 nearest neighbors instead of obtaining the determining set by linear interpolation.

Remark 5. *Geometry shaders have been introduced in Shader Model 3.0 as a new pipeline stage in between the vertex and the fragment shaders. At first glance, geometry shaders would help to remove some of the redundancies in the vertex processing of our cell projection based algorithm, for instance the calculation of barycentric coordinates. One of the problems of geometry shaders is that performance is affected by the number of variables which are passed to the fragment stage. According to [NVI08b] geometry shader performance drops by 50% if this number is higher than 20. With nine float values per vertex (coordinates and barycentrics) and three visible triangles encoded as a strip of five vertices we have 45 float values per tetrahedron. Hence, usage of geometry shaders does not improve performance of our rendering approach.*

Remark 6. *Note that the depth of a fragment needs to be adjusted only if clipping with arbitrary surfaces such as planes, curves or triangle meshes is required. See Figure 6.23 for an example. In this case, a fragment's depth d is given by*

$$d = z_{far}/(z_{far} - z_{near}) + z_{far} \cdot z_{near}/(z_{near} - z_{far})/||\widetilde{\mathbf{x}}||,$$

where $||\widetilde{\mathbf{x}}||$ is the length of the vector defined by the intersection point $\mathbf{x}$ transformed into eye space.

Chapter 7

Surface Reconstruction from Unstructured Points

In the previous chapters, we considered the reconstruction and visualization of smooth surfaces from discrete data on regular grids. In this chapter, we address the problem of finding a surface approximation from a different data source, namely unstructured 3D point sets. Typically, unstructured point sets or *point clouds* are generated by a dense sampling of real-world object contours by using 3D laser range scanners. Point clouds have various applications, e.g., quality control and reverse engineering of prefabricated parts in production processes, the digitalization of artifacts in archaeology and forensics, or archival storage of objects of high cultural relevance such as buildings and statues.

Besides direct visualization by splatting [ZPvBG02] or ray casting [AA03, SGS06] various approaches that reconstruct a continuous surface from its discrete representation as point sets are known. A common and versatile continuous surface representation are polygonal meshes. They are efficient in memory consumption and can be processed with high performance on current graphics hardware. Therefore, the construction of polygonal meshes has received a lot of attention and a substantial amount of literature, algorithms and techniques on this topic has been published. The generation of a mesh aims at finding a partition of the surface domain from an unstructured point set $\mathbf{P}$. The elements of the partition are typically triangles or quads. The notion of *quality* of the resulting mesh has several meanings and can refer to the sampling rate (i.e., triangle sizes adapt to local surface characteristics like curvature), regularity (i.e., most vertices have degree 6), or shape (i.e., the relation between inscribed and circumscribed circles) of the elements, see Figure 7.1. Many post-processing algorithms such as subdivision or mesh compression, and numerical simulations profit from nicely shaped triangles and regularity of the meshes.

To obtain a triangulation from some surface representation one can choose between several approaches. We give a brief overview of this related work in Section 7.1. A certain type of method depends on a projection which maps newly created vertices onto the surface. These projective methods are especially well suited for advancing front algorithms which allow for generation of high-quality meshes that adapt to local surface curvature. A very popular projective method is the Moving Least Squares (MLS) approach for surface reconstruction. The power of MLS surfaces is the ability to naturally cope with input noise in the data.

However, we show in this chapter that the MLS approach has some deficiencies and

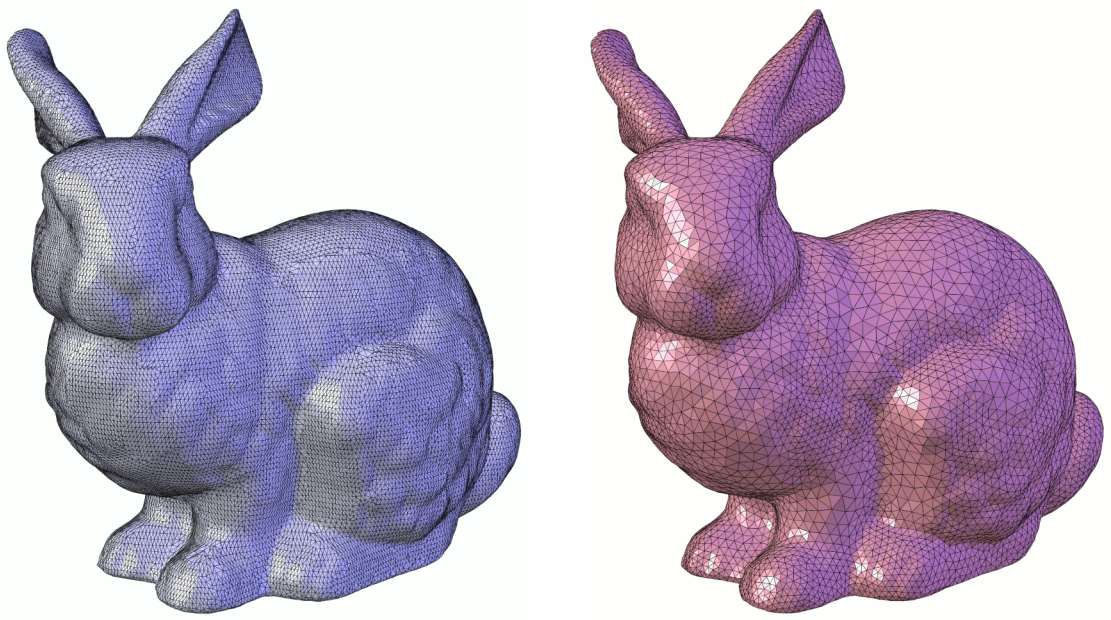

Figure 7.1: The *Bunny* model. *Left*: A typical mesh reconstructed from measured points (35k vertices). *Right*: The same model in a high-quality triangulation (about 10k vertices).

can lead to problems in the computation of both the tangent plane and the polynomial approximation of the surface. Additionally, points that are beyond the vicinity of the surface are not guaranteed to be correctly projected with MLS. We propose a new projection method that does not involve a non-linear optimization or similar problem. The approximated points are iteratively collected compromising connectivity information. We enhance the orientation of the local coordinate system to further improve the method. Furthermore, in our method we can project points which have arbitrary distance to the surface. In a first step we find a local parameter domain; this is closely related to the Cocone [ACDL00] algorithm. In a second step we approximate the surface by a polynomial. Unlike MLS, we do not use a fixed polynomial degree but adapt the degree depending on the location of the points to be approximated. Some scanning device estimate a normal for each point, but often the normals are unreliable due to the lighting conditions (i.e., shadows, reflections) in the scanning process. Our method works very well in these cases, since we do not rely on precomputed normals of the points. The results confirm that our method is more robust and also accelerates triangulation due to a preprocessing step that needs to be done only once per data set. An example of the triangulations obtained by our method is shown in Figure 7.2.

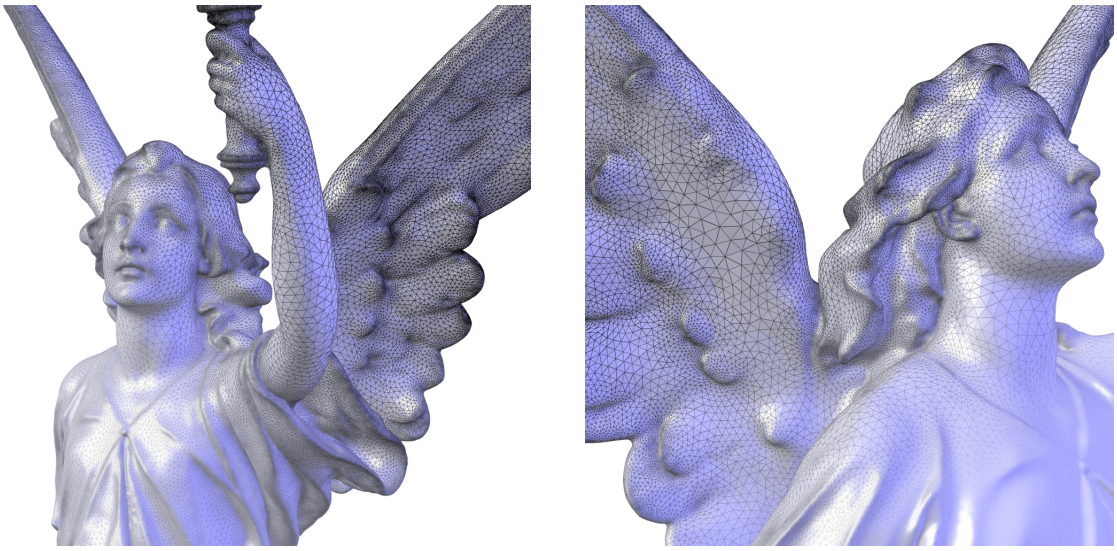

Figure 7.2: Triangulation of the *Lucy* model using our new projection method.

7.1 Related Work

In surface reconstruction methods from measured data we distinguish between approaches that consider the structure of the acquired data, such as range images, and methods based on unstructured points. Popular representatives of the first class are Turk and Levoy's zippered range images [TL94a], and volumetric range image processing (VRIP) by Curless and Levoy [CL96b]. Here, we concentrate on reconstruction methods from unstructured point sets.

Some authors proposed to construct an implicit function from an unstructured point set $\mathbf{P}$ that partitions the space into interior and exterior [HDD*92, Kol05, KBH06]. The surface is then defined by the function's kernel [BW97]. As a result these functions nicely close unintended gaps in the point cloud but are not able to reconstruct surfaces with borders. The final surface is often extracted as a triangular mesh by applying marching cubes. However, this usually leads to aliasing or oversampling and badly shaped triangles. A remeshing algorithm can be applied to the mesh obtained from marching cubes to improve the quality in a post process [AdVDI03, SG03, FZ09, FAKG10]. One problem here is that most remeshing algorithms are based on a reparameterization which is expensive to compute and can be non-trivial for very large models [SFS05].

Computational geometry approaches such as Alpha Shapes [EM92, BB97], (Power) Crust [ABK98, ACK01, ACK00] and Cocone [ACDL00, DG01, DGH01] usually apply a Delaunay triangulation as a basic step. Although these methods often provide theoretical reconstruction guarantees, the number of generated vertices and triangles cannot be directly controlled but is related to the cardinality of $\mathbf{P}$, and triangulations are not free from overlaps. Another problem is the direct reflection of measurement noise: In its

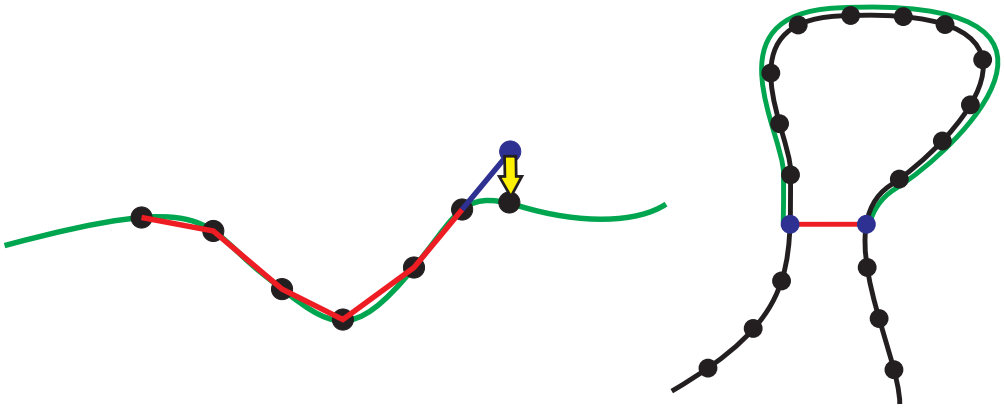

Figure 7.3: *Left*: 2D illustration of the general principle of projective methods for triangle mesh generation. The current front (red) is extended by predicting a new point (blue) which gets projected onto the surface (green). *Right*: Riemannian vs. Euclidean distance: the blue points have a small Euclidean (red), but large Riemannian distance (green) on the surface and should not be considered as neighboring points in the weighting.

basic form, the Cocone is only suitable for noise-free surfaces. An extensions of the Cocone for noisy point sets has been given by Dey and Goswami [DG04].

Projective methods like advancing front successively construct a triangulation from the border of the triangulated domain (the *front*), starting with an initial triangle. The general principle has been proposed by Hartmann [Har98] to obtain high-quality triangulations of smooth implicit functions. New points are predicted by some heuristic, e.g, by mirroring the vertex opposite to the current front edge, and then projected onto a locally reconstructed surface, see Figure 7.3, left, for a 2D illustration. Hartmann proposed the topological operations *merge* and *split* to handle the special cases of two meeting fronts or a front crossing itself. This method has been later extended such that edge lengths adapt to local curvature and has been applied to real-world data [KS01, SFS05, SSS06a]. The projection step onto the locally reconstructed surface can be done with, e.g., the MLS method [Lev98, ABCO*01, ABCO*03]. In a first step the MLS method calculates the tangent plane of the surface near the point that is to be projected. Next, the surface is locally approximated with a polynomial in the parameter domain defined by the tangent plane. Unfortunately, finding this plane involves a non-linear optimization problem that can only be solved numerically, e.g., by using conjugate gradients. Scheidegger et al. [SFS05] propose a heuristic to find an appropriate initial value for the gradient descend solver. To be able to robustly solve the involved optimization problem the point to be projected must be within the vicinity of $\mathbf{P}$. Furthermore, as pointed out by [AK04], the solution of the optimization problem does not always lead to a tangent plane of the surface but can significantly differ from the correct plane. Calculating the polynomial can lead to similar problems if different regions of the surface approach each other. As a result, points whose Euclidean distance is small are taken into consideration even though their Riemannian distance is large.

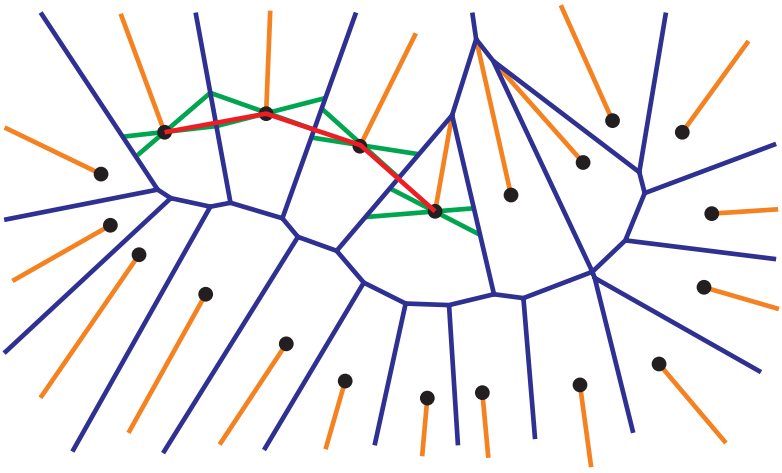

Figure 7.4: 2D illustration of the Cocone algorithm for surface reconstruction. First, the poles (orange) are determined for each Voronoi cell (blue). The co-cones are shown in green. Two neighboring points are connected if their co-cones meet at a common edge of their corresponding Voronoi cells.

We aim at curing the situation described above with an improved projection method that does not involve solving a non-linear optimization problem. The method makes use of connectivity information and uses a variable degree for the polynomials to improve robustness of the polynomial approximations.

7.2 A New Projection Method for Point Set Surfaces

Many problems of finding the local parameter planes and polynomial approximations are caused by missing connectivity information of the points in $\mathbf{P}$. Points having a small Euclidean, but large Riemannian distance might be wrongly considered as neighboring points, see Figure 7.3, right. With the connectivity information it is easier to create the tangent plane and an improved polynomial free of points with high Riemannian distances. Our projection is based on a precalculation step where we establish connectivity information to approximate the Riemannian distances of the points. This precalculation is independent of the triangulation parameters and needs to be done only once per data set. This results in an efficient projection method that makes use of the precalculated data.

First, in our precalculation step, we reconstruct the surface locally at each point $\mathbf{p}_i \in \mathbf{P}$ with bivariate polynomials of variable degree. We do so by applying weighted least squares in a local coordinate system (LCS). To find a good approximation of the normal and to correctly weight the points by approximate Riemannian distance, we establish inter-point connectivity using the Cocone algorithm. Roughly speaking, this algorithm creates a triangulation from double cones (the *co-cones*) centered at the $\mathbf{p}_i$ and which are aligned orthogonally with the *poles* of each Voronoi cell, i.e., the Voronoi vertex with the farthest distance to $\mathbf{p}_i$. See Figure 7.4 for an illustration of the basic

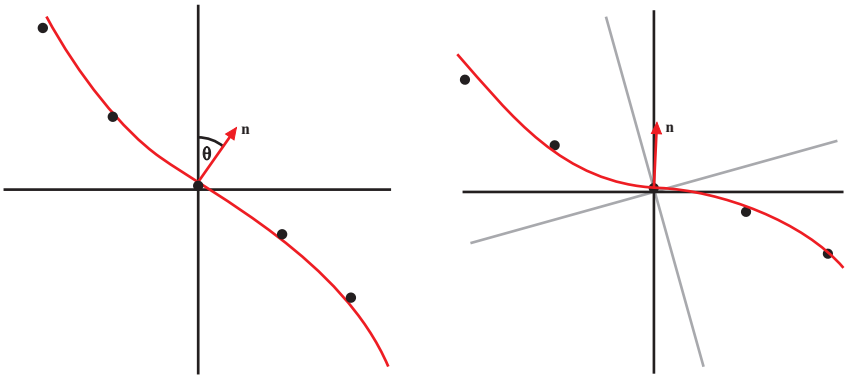

Figure 7.5: *Left*: the angle θ between the z-axis of the local coordinate system (LCS) and the surface normal $\mathbf{n}$ of $p(\mathbf{x})$ is too large. *Right*: a new LCS is obtained iteratively by the bisection of the z-axis and $\mathbf{n}$.

principle. To actually project a point $\mathbf{r}$ onto the surface, we find the closest point $\mathbf{p}_i \in \mathbf{P}$ and project $\mathbf{r}$ onto the polynomial surface of $\mathbf{p}_i$. In order to compensate for single outliers in the projections, we consider the polynomial surfaces of neighbors of $\mathbf{p}_i$ and use the component-wise median of the projections.

The Cocone algorithm guarantees that points are connected topologically correct if $\mathbf{P}$ is an ϵ-sample with $\epsilon \leq 0.08$, which means that for each point $\mathbf{s}$ on the surface, there exists a point $\mathbf{p} \in \mathbf{P}$ which is closer to $\mathbf{s}$ than ϵ times the distance from the medial axis to $\mathbf{s}$. This guarantee is also valid for surface reconstructions based on our projection method. MLS guarantees correct reconstruction for $\epsilon \leq 0.01$ which is only valid if point normals are associated with $\mathbf{P}$. In practice the Cocone performs well with ϵ up to 0.5 [ABK98]. Point normals are approximated as a byproduct and can be used to initially align the LCS of the polynomial in the next step.

Once inter-point connectivity is calculated we transform the points of $\mathbf{P}$ into the LCS centered at $\mathbf{p}_i$ and the z-axis is aligned with the normal approximated by the Cocone algorithm. We approximate the surface locally at each point $\mathbf{p}_i \in \mathbf{P}$ with a bivariate polynomial $p(\mathbf{x}) \in \mathcal{P}_q$ of variable degree q which also provides a surface normal by $\frac{\partial p}{\partial x} \times \frac{\partial p}{\partial y}$ and curvature information. We use the normal of the polynomial to improve the orientation of the LCS: if the angle between the z-axis and the surface normal exceeds a specified threshold, we obtain a new z-axis by bisection, see Figure 7.5, and repeat the polynomial approximation based on the new LCS. We refer to the transformation into the LCS of a point $\mathbf{p}_j \in \mathbf{P}$ as $(\mathbf{x}_j, f_j)$ with $\mathbf{x}_j = (x_j, y_j)^\mathsf{T} \in \mathbb{R}^2$ and $f_j \in \mathbb{R}$. To find the local surface approximating polynomial we minimize the sum of weighted error squares

$$e = \sum_{j=1}^{N} (p(\mathbf{x}_j) - f_j)^2 \cdot \omega \left(\widetilde{R}(\mathbf{p}_i, \mathbf{p}_j) \right), \tag{7.1}$$

with the approximate Riemannian distance $\widetilde{R}$ and the weighting function $\omega(d)$. As in

similar applications we set $\omega(d)$ to

$$\omega(d) = \mathrm{e}^{-\frac{d^2}{h^2}} \tag{7.2}$$

whereas h can be interpreted as smoothing factor. By default h should be set to the local sample spacing at $\mathbf{p}_i$ but can be increased to smooth out measurement noise [ABCO*03].

We refer to the polynomial basis of $p(\mathbf{x})$ depending on the degree q as $\mathbf{b}(\mathbf{x}) = (1, x, y, x^2, xy, y^2, \ldots)$ and to the polynomial coefficients as $\mathbf{c} = (c_1, c_2, c_3, c_4, c_5, c_6, \ldots)$. Therefore $p(\mathbf{x}) = \mathbf{c} \cdot \mathbf{b}(\mathbf{x})^\mathsf{T}$ and we can rewrite Equation (7.1) as

$$e = \sum_{j=1}^{N} \left(\mathbf{c} \cdot \mathbf{b}(\mathbf{x})^\mathsf{T} - f_j \right)^2 \cdot \omega \left(\widetilde{R}(\mathbf{p}_i, \mathbf{p}_j) \right). \tag{7.3}$$

To find the coefficients minimizing the error square sum we set the partial derivatives $\left(\frac{\partial e}{\partial c_1}, \frac{\partial e}{\partial c_2}, \ldots \right)$ to zero which leads to the linear system of equations

$$\mathbf{A}\mathbf{c} = \mathbf{d} \tag{7.4}$$

with

$$\mathbf{A} = \sum_{j=1}^{N} \mathbf{b}(\mathbf{x}_j)^\mathsf{T} \mathbf{b}(\mathbf{x}_j) \cdot \omega \left(\widetilde{R}(\mathbf{p}_i, \mathbf{p}_j) \right),$$
$$\mathbf{d} = \sum_{j=1}^{N} \mathbf{b}(\mathbf{x}_j)^\mathsf{T} f_j \cdot \omega \left(\widetilde{R}(\mathbf{p}_i, \mathbf{p}_j) \right).$$

Theoretically, every point is taken into account in order to find the minimum error sum. However, since Equation (7.2) decays fast, in practice it is sufficient to only consider points in a certain distance to $\mathbf{p}_i$. The error in a distance of $4h$ is almost negligible and for $6h$ it is around double precision.

We approximate the Riemannian distance with the Euclidean distance but only consider candidate points $\mathbf{p}_j$ if there exists a path (given through inter-point connectivity) from $\mathbf{p}_i$ to $\mathbf{p}_j$ and there is no point $\mathbf{p}_k$ on the path with $\|\mathbf{p}_k - \mathbf{p}_i\| > r_c$, where r_c is the maximum point distance. To ensure a good polynomial approximation and to prevent the triangulation from overlapping, points are iterated in a breadth-first search, projected onto the xy-plane and only collected if the projection does not lie within the convex hull of already collected points, see Figure 7.6 for a 2D example.

Solely solving Equation (7.4) does not automatically lead to a good local surface approximation. As pointed out in [HZDS01, DZ04], a surface approximation can fail if all $\mathbf{x}_j$ in the local coordinate system are near an algebraic surface of same or lower degree than q. In these cases, a small variation of the points can lead to completely different polynomials. An example is given in Figure 7.7. We can use the matrix condition $\kappa(\mathbf{A})$ as criterion to detect if the collected points are near an algebraic surface. However, high matrix conditions are not only caused by points in the vicinity of an algebraic curve but also from an inappropriate scaling. To avoid high κ we scale the points of $\mathbf{P}$ when

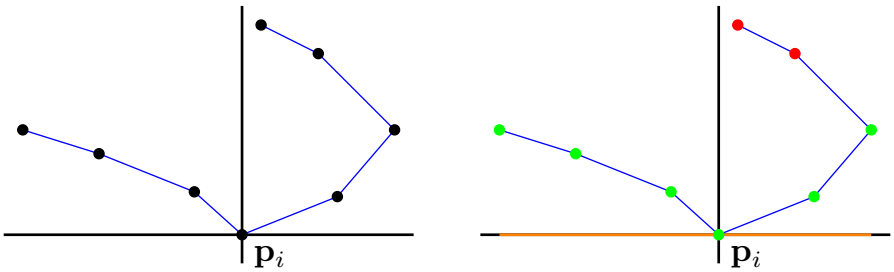

Figure 7.6: *Left*: overlapping points in the coordinate system prevent a good approximation. *Right*: (green) points not leading to overlaps are collected; (red) points whose projections intersect the convex hull (orange) are rejected.

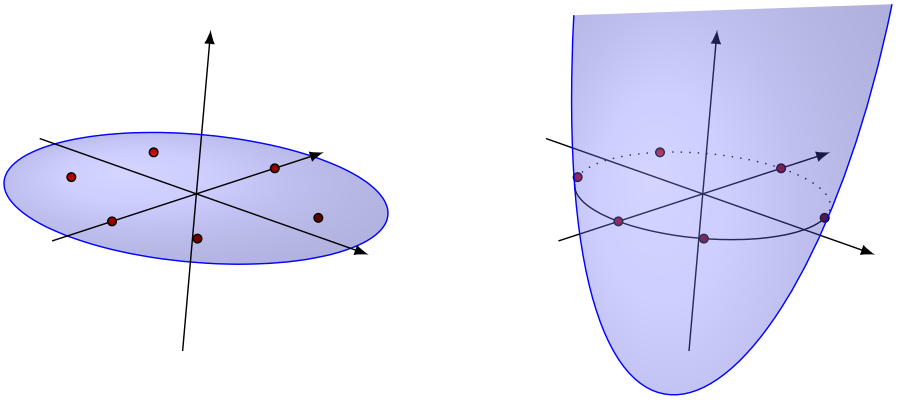

Figure 7.7: Points are near an algebraic surface. They might lie on a plane (*left*) as well as on a paraboloid (*right*).

transforming them to the LCS. We observed that κ is lowest if the average point spacing is approximately 1. The matrix condition is then a reliable criterion to check if the points lie near an algebraic surface.

We build matrix $\mathbf{A}$ according to Equation (7.4) starting with an initial degree $q_{\max}$ and check if $\kappa(\mathbf{A})$ exceeds a threshold (which is independent of $\mathbf{P}$). In this case, we reduce the polynomial degree by one and rebuild $\mathbf{A}$. We repeat this step until the condition is below the threshold or we reach a degree of 1. We note that reasonable matrix conditions lie between 200 and 400 and we can set $q_{\max}$ to around 5. The above strategy allows us to use polynomials of high degree that nicely adapt to local points without the side-effects various methods not considering matrix conditions suffer from.

7.3 Results

We have given a new projection method which is based on stable and efficient algorithms and does not rely on non-linear or similar optimization problems. By using the Cocone

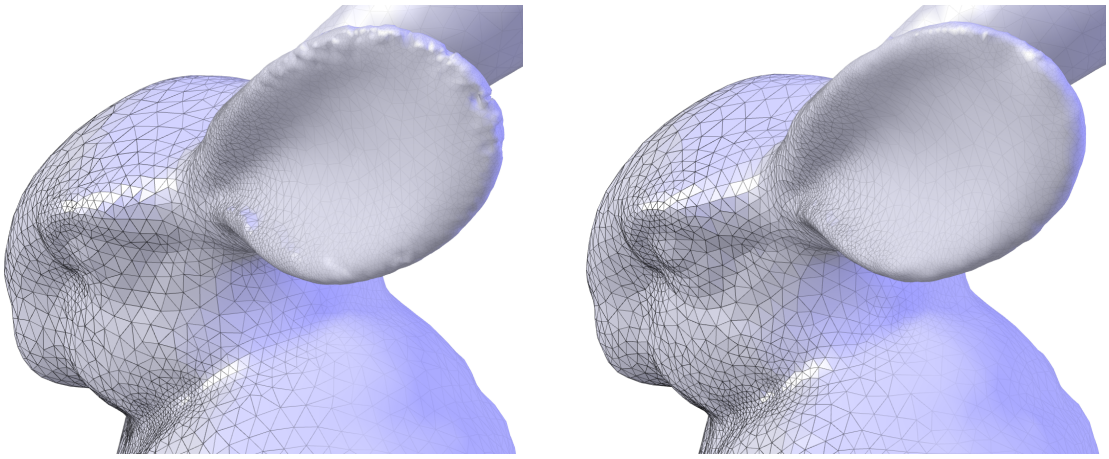

Figure 7.8: The *Bunny* ear. *Left*: MLS reconstruction. *Right*: our method.

algorithm in a preprocessing step we can estimate normals and Riemannian distance of the points on the surface. The reconstruction guarantees of the Cocone are significantly better than for MLS which is also valid for our projection method. Numerical problems in calculating the local surface approximations have been reduced by considering matrix condition and adapting the degree of the bivariate polynomials.

We implemented a mesh generation tool *CloudMesh* [Uhr08] with an advancing front algorithm based on our projection method. Triangulations using MLS were performed with *Afront* [SSFS07]. We provide results of our experiments with a collection of standard data sets, namely the Stanford models *Lucy*, *Bunny* and *Dragon* (see Figures 7.2, 7.8 and 7.9).

Data Set	*Bunny*	*Lucy*	*Dragon*
Points	34 834	262 909	435 545
Connectivity	0:32	4:15	7:26
Reconstruction	7:08	50:14	84:52
Triangulation	1:55	29:55	18:00
Vertices	24 802	442 422	198 157
Triangles	49 421	881 725	389 709

Table 7.1: Timings and results for various data sets.

Figure 7.8 shows that our approach is more robust and improves the triangulation results especially in regions of high curvature. A close-up of the dragon head (see Figure 7.9) confirms that graduation in triangle size is more plausible due to polynomial reconstruction with variable degree and the fact that larger edge lengths can be chosen than with MLS. The ratio of the diameters of the incircle and circumcircle for an equilateral triangle is 0.5. By construction, advancing front algorithms naturally create

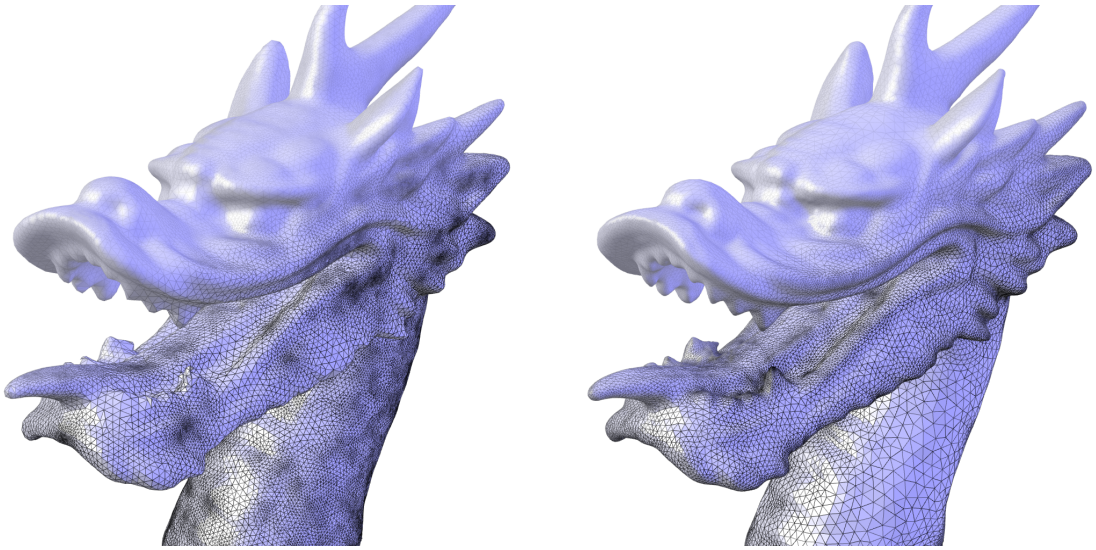

Figure 7.9: The *Dragon* head. *Left*: MLS reconstruction. *Right*: our method.

nearly equilateral triangles in the majority of cases. The meshes reconstructed in our application have an average ratio of 0.46 to 0.47, which is comparable to the Afront implementation. Further, in our tests we have 74 to 76% degree six vertices, which is slightly better than Afront.

The preprocessing step consists of establishing connectivity and the polynomial surface reconstruction. Here the computation time linearly depends on the number of points in $\mathbf{P}$. We note that the preprocessing step is easily parallelizable and has to be done only once for each data set since it is independent of any triangulation parameters. Triangulation times depend on the number of points as well as generated triangles. As a result, objects with many highly curved areas have increased triangulation time, see Table 7.1. All timings were done on a single core 2.4 GHz AMD CPU. The given triangulation times are not optimal, since intermediate results are visualized during the triangulation process.

Chapter 8

Summary and Discussion

In this thesis, we have investigated trivariate splines on tetrahedral partitions for data approximation, where we focused on the practical aspects of visualizing isosurfaces from real-world data on current GPUs. Furthermore, we presented new theoretical results for this type of splines, where we solved the problem of finding a local data approximation method by piecewise quadratic polynomials, which is globally C^1. Finally, we presented a new projection scheme for the reconstruction of 3D surfaces from unstructured data points. In the following, we summarize our main results and give an outlook on further research.

Trivariate Splines for Data Approximation

Smooth trivariate splines in B-form have several advantageous features which makes them a very good choice for data approximation on volumetric grids. Since the polynomial pieces are given in the Bernstein-Bézier form, we can rely on a large set of well-approved tools for the evaluation of the polynomials as well as for the structural analysis of associated spline spaces. The convex hull property of the B-form allows for quick culling tests of tetrahedra which do not contribute to the surface. Evaluation of the polynomials using de Casteljau's algorithm is numerical very stable, since it is based on convex combinations of the polynomial coefficients. This can be compared to, e.g., trivariate box splines, which are non-trivial to evaluate in a numerical stable way [dB93]. Blossoming as a generalization of de Casteljau's algorithm can be used in ray casting to obtain a univariate representation of a trivariate polynomial along the viewing ray. Intermediate results can be reused for efficient gradient evaluation. For continuity between neighboring polynomials, simple relations of the involved coefficients have to be fulfilled, see Section 4.4 and 5.2. On the other hand, smooth trivariate splines are very complex spaces, since smoothness has to be established across all faces of the tetrahedral partition, while at the same time, we want to approximate the given data with a bounded error.

Various approximation and interpolation methods based on splines on tetrahedral partitions are known, see Chapter 4. We use quasi-interpolating operators which have significant advantages compared to classical finite element or Lagrange interpolation methods. Our quasi-interpolation methods do not rely on splits of tetrahedra, keeping the total number of tetrahedra low. Derivatives do not have to be known or estimated at the grid points or across edges, and no interpolation points, which might not lie

on vertices of the volumetric grid, have to be defined. Further, the computational complexity of the quasi-interpolating methods is linear in the number of data points, and the methods are local and stable, since small and local changes of the data have only small and local influence on the splines.

Today, only a few smooth and local trivariate quasi-interpolating schemes with guaranteed approximation properties and low polynomial degree are known. Examples are the quadratic super splines and cubic C^1-splines on type-6 tetrahedral partitions which we used for high-quality visualizations of isosurfaces from volume data. The practicability of these splines derives from the speed and stability of working with polynomials in B-form, and the efficiency of computing the B-form coefficients, which are linear combinations of nearby function values. The basic challenges in these approaches are to find a reasonable simple tetrahedral partition and an evenly distributed determining set of the splines. Furthermore, we have to assign the coefficients of the determining set from the data in a local neighborhood such that the spline is stable and has adequate approximation properties, which is especially difficult for splines with a low polynomial degree. In addititon, the tetrahedral partition has to be chosen with care, because the approximation properties of the splines depend on the smallest angle of the triangles in the partition.

Splines on Truncated Octahedral Partitions

We introduced a new approximation scheme in Chapter 5 generating piecewise C^1 quadratics using a specific triangulation of $\mathbb{R}^3$. We further analyzed the convergence properties of this new scheme and applied it for isosurface visualization from volume data. Further, the nature of our quadratic C^1-splines on truncated octahedral partitions motivates the direct usage of data on the BCC lattice, which is a superior sampling grid compared to the cubic lattice because equal approximation quality can be achieved with fewer data samples. By adapting our approximation scheme to work with data on BCC lattices, we are confident to achieve an optimal order of approximation. In a next step, the structure of C^2-splines on truncated octahedral partitions should be analyzed. First investigations on this topic have shown that local quasi-interpolation operators with an expected degree of four could be constructed. To our knowledge, this would be the first C^2-smooth spline approximation scheme with a lower degree than five.

Real-Time Visualization by Ray Casting

Quasi-interpolating trivariate splines of low polynomial degree are also well-suited for real-time visualization by ray casting, which is the most flexible method for volume visualization and which also gives the highest visual quality, see Section 2.2. The low polynomial degree allows for analytic, or stable and efficient numeric root finding for ray intersections. In contrast to this, most methods based on tensor-product splines rely on interval refinement methods, which can be difficult near silhouettes.

Artifacts caused by trilinear interpolation, such as blocky silhouettes, are significantly reduced, while the total degree of the polynomial pieces does not exceed that of trilinear

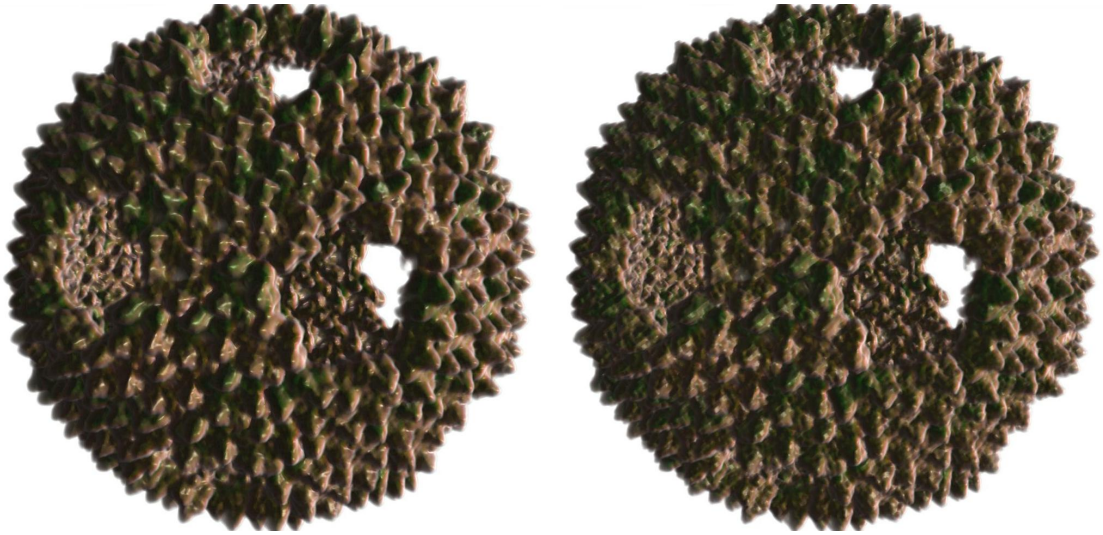

Figure 8.1: Virus Bacteriophage PRD1 model 1HB5 (128^3 voxels). *Left*: quadratic super splines. *Right*: trilinear interpolation.

interpolation. Compared with, e.g., triquadratic or tricubic tensor product splines, which lead to piecewise polynomials of total degree six, or nine, respectively, our splines have significantly lower overhead for the evaluation of the polynomials and its derivatives. Gradients, needed for illumination, are directly available from the evaluation of the polynomials. In contrast to this, trilinear tensor products often rely on finite differences for gradient estimation. Besides artifacts caused by the resulting non-smooth normals, computing finite differences is more expensive than the evaluation of the trilinear model itself. Small data stencils are always preferable for real-time renderings of the models, since less data values have to be fetched from memory. Features are also better preserved, since a smaller number of coefficients has influence in the weightings. Note that the data stencils in our methods do not exceed the size of 27 neighboring values, which holds also true for cubics. In contrast to this, tricubic tensor products are based on a 64 neighborhood. The situation is even worse in the case of trivariate box splines [EM06, FEVM10], where a $5 \times 5 \times 5$ neighborhood is used.

Hybrid Cell Projection / Ray Casting

At the time when the first rendering method based on trivariate splines was published in 2005 [NRSZ05], it has been a complete offline rendering method. Since then, the increasing flexibility and performance of current GPUs has motivated several schemes for the rendering of B-form polynomials on tetrahedral partitions. We have shown an improved cell projection scheme for Bézier tetrahedra, which outperforms similar schemes [LB06, SWBG06, SGS06, KOR08] both in terms of rendering performance, as well as memory efficiency. Spline structure needs to be taken into account in order to reduce the memory demands for storing the spline coefficients. We examined several schemes, varying from precomputed subsets of coefficients, from which the remaining

coefficients can be computed on-the-fly by repeated averaging, up to a complete on-the-fly computation of the splines on the GPU. Further, with the exception of [KOR08], previous methods do not consider the advantages of uniform tetrahedral partitions for the storage of the bounding geometries of the piecewise polynomials. In contrast to this, we use an implicit scheme for geometry encoding in combination with instancing on the GPU for an efficient rendering approach.

Cell-projection based methods are simple to implement and fit nicely into the general rendering pipeline. For instance, texturing or clipping with arbitrary surfaces can be done with high quality. On the other hand, we have shown the limitations of cell projection for a very large number of tetrahedra. In these cases, high overdraw occurs and many triangles are smaller than the optimal batch size for fragment processing on the GPU. Further, the number of vertices soon outperforms the number of visible fragments and vertex processing becomes a bottleneck. Thus, cell projection is a good choice for relatively small surfaces consisting of up to several hundreds of thousands of polynomials. On the other hand, real-world volume data arising from modern scanning devices can be very large. Today volumes of sizes up to 512^3 voxels are common in medical praxis, resulting in isosurface consisting of several ten million polynomials. This has motivated the development of a geometry free, *image based* ray casting for trivariate splines.

Image Based Ray Casting

In our pure image based ray casting for trivariate splines, for each pixel in the image plane we traverse a ray throughout the volume. Diverging threads are hereby avoided by a fast empty space skipping scheme, as well as two rendering passes. The first rendering pass sorts out those rays which are guaranteed to miss the surface. The remaining rays are adjusted such that the start and end points of the rays coincide with the entry and exit points of the first and last contributing tetrahedron, respectively, along the ray. In the second pass we only consider rays with a potential surface intersection. For each data cube the ray passes through, we calculate the determining set of the cube, where we profit from the small data stencils. Next, we traverse the tetrahedra within the cube hit by the ray. The remaining B-coefficients for each tetrahedron are calculated by simple averages of the coefficients from the determining set. The uniformity of the tetrahedral partition allows for a quick traversal with only a few number of conditionals. Empty space skipping is based on octrees, which allows for an efficient traversal of empty cells. Improved traversal methods, for example anisotropic chessboard distance traversal [vK00], can be used, but at the cost of a more complex preprocessing when the isosurface changes.

In the image based ray casting, we relief the GPU from the geometry processing bottleneck, which is directly reflected by the achieved frame rates for very large numbers of tetrahedra. Further, spline coefficients are always computed on-the-fly on the GPU from the volume data. We thus do not have to inflate the data prior to rendering to encode the spline coefficients. It is straightforward to extend our direct ray casting to deal with transparent isosurfaces and full volume rendering. The image based approach was both developed in the OpenGL graphics pipeline, as well as using CUDA, a general

purpose programming language and toolkit beyond the graphics pipeline. The OpenGL approach achieves a speedup of almost an order of magnitude compared to our fastest cell projection: for isosurfaces with more than ten million tetrahedra we have nearly ten frames per second for image based ray casting compared to less than one frame with cell projection. The speedup of using CUDA is even more impressive: here we have about 30 frames per second using the same hardware. The massive speedup of the CUDA implementation is based on using persistent threads, i.e., a pool of threads, where a thread with a fast path does not need to wait for slower threads in the same batch, which results in a significantly better utilization of the hardware.

Conclusion and Outlook

We achieved the main goal of our work, which is to apply trivariate splines on uniform tetrahedral partitions for real-time reconstruction and visualization of isosurfaces from real-world volume data. Our kernels for isosurface visualization scale well with increasing performance of GPUs. First tests with newest generation GPUs, i.e., the NVidia G400 architecture, gave us a performance gain of more than 100%. The speedup is based on the increasing number of processing units as well as a higher register count, allowing to process more threads in parallel. Further, our approach benefits from an improved caching of global and texture memory.

Our rendering schemes can also be applied to alternative splines on tetrahedral partitions. Examples are the quintic C^1-splines on type-4 tetrahedral partitions [SS04], and the cubic C^0-splines based on MLS approximation [KOR08]. However, our quasi-interpolating operators have a number of advantages compared to the afore mentioned methods. The quintic polynomials used in [SS04] have significantly more Bernstein coefficients on each tetrahedron than quadratic and cubic polynomials, namely 56 instead of 10, and 20, respectively. Quintic splines have a higher overhead for the computation and storage of the splines, as well as for the evaluation of the polynomials. The high register use on the GPU makes these schemes impractical on current graphics hardware. Different from MLS splines [KOR08], our quasi-interpolants are directly available from appropriate weightings in a small local neighborhood. This allows for an on-the-fly computation of the splines directly on the GPU and a significantly lower memory footprint The main advantage of quintic Lagrange interpolants and cubic MLS splines are simpler tetrahedral partitions (see Figure 4.2), which directly results in a smaller number of tetrahedra. The higher number of tetrahedra in the type-6 partition, see Section 4.4, and our tetrahedral partition of truncated octahedra, see Chapter 5, is a significant burden for rendering schemes based on cell projection. However, we showed that a direct and geometry free ray casting method for isosurfacing resolves the bottleneck of a high tetrahedron count. This would also motivate to consider cubic C^1 Lagrange interpolation methods that rely on splits of (certain) tetrahedra in up to 24 subtetrahedra, e.g., [NSZ05, HNSZ08, HNSZ09], for real-time data approximation and visualization.

The fast development of graphics hardware may allow us to consider splines with higher degree than cubics for interactive reconstruction and visualization in the near future. Another interesting field is the development of efficient GPU approaches for scattered

data interpolation and approximation based on trivariate splines. In this context, we are confident to achieve better performance than comparable approaches, such as MLS volume ray casting by Ledergerber et al. [LGM*08].

A New Projection for Point Set Surfaces

In Chapter 7 we covered the reconstruction of surfaces from unstructured points where we developed a new projection method for the generation of high-quality triangle meshes in an advancing front approach. Among the various reconstruction techniques from unstructured points, this is one of the few methods that directly generates triangle meshes which meet certain quality criteria, such as triangle shape (almost equilateral triangles) and vertex degree (most vertices have degree six). Further, the detail level, i.e., average edge length, can be controlled by a few user defined parameters and the triangulation adapts to local surface curvature by using a guidance field.

Advancing front algorithms rely on a projection step which is usually based on finding a local surface frame by MLS approximation. To do this, a non-linear optimization problem has to be solved which can be done only numerically and the solution depends on finding suitable starting values for a gradient descent solver. Further, only points in a close vicinity of the surface are guaranteed to be projected correctly. To circumvent these problems, we propose an alternative approach to improve the stability of advancing front algorithms for triangle mesh generation where we do not need to solve non-linear optimization problems. Points to be projected can be within arbitrary distance to the surface, which implies that we can generate coarser triangulations than in the MLS approach.

In a preprocess, we first establish inter-point connectivity by using the Cocone algorithm, which is based on a Delaunay tetrahedralization of the points. The Cocone has significantly better reconstruction guarantees than MLS. The inter-point connectivity allows us to approximate the Riemannian distance of the points on the surface. This improves the stability of the polynomial approximations of the surface in a local neighborhood, since points with small Euclidean distance, but with large distance on the surface, are not considered in the weightings. Further, the normals obtained from the Cocone can be used to improve the orientation of the local coordinate frame. This preprocess is straightforward to parallelize and has to be done only once for each data set because it is independent of triangulation parameters.

The surface is then locally approximated by bivariate polynomials of varying degree obtained from a least squares approximation of neighboring points weighted by approximate Riemannian distance. If the points are near an algebraic surface of same or lower degree, small changes of the points can lead to completely different polynomials. To improve the stability of the approximations, we thus reduce the polynomial degree if the matrix condition is low. This allows us to use high degree polynomials that nicely adapt to local surface structure while the side effects methods not considering matrix conditions suffer from are avoided. Further, we present an improved strategy to prevent fronts from overlapping based on constructing the convex hull of the relevant points.

Future Challenges of Projection Methods

One of the open problem of advancing front algorithms is the speed of the triangle mesh generation. Triangles cannot be considered independently from each other, since we have to account for topological events such as meeting fronts. A general problem of surface reconstruction methods from unorganized points is a deficient sampling in complex or occluded regions of the surface, where the scanning process fails to produce enough points for a topological correct reconstruction. We significantly improved the stability of advancing front in these cases, but there are still situations where the surface is not reconstructed correctly. With an automatic detection of these regions, one could ignore them in a first step in order to prevent the propagation of a wrongly reconstructed area over the whole surface. Closely related to this problem is the correct recognition of borders, as it is difficult to distinguish them with a gap in the sampling. Finally, while advancing front works very well for organic objects, an open problem is to find a natural way to deal with sharp features as they occur in machinery parts, for instance. Besides starting separate fronts from sharp edges, no extension of advancing front for sharp features has been proposed yet.

Appendix A

Publications

[1] Fuhrmann, S., Ackermann, J., Kalbe, T., and Goesele, M. Direct resampling for isotropic surface remeshing. In *Proceedings Vision, Modeling and Visualization (VMV)* (2010).

[2] Kalbe, T., Fuhrmann, S., Uhrig, S., Zeilfelder, F., and Kuijper, A. A new projection method for point set surfaces. In *Eurographics 2009 - Annex: Tutorials, State of the Art Reports, Short Papers, Medical Prize, Education Papers, Areas Papers, EG Workshop on Natural Phenomena* (2009), pp. 77–80.

[3] Kalbe, T., Koch, T., and Goesele, M. High-quality rendering of interactively varying isosurfaces with cubic trivariate C^1-splines. High Performance Graphics (HPG), New Orleans, USA, 2009. Poster Presentation.

[4] Kalbe, T., Koch, T., and Goesele, M. High-quality rendering of varying isosurfaces with cubic trivariate C^1-continuous splines. In *Proceedings of 5th International Symposium on Visual Computing (ISVC), Las Vegas, USA* (2009), pp. 596–607.

[5] Kalbe, T., Tekušová, T., Schreck, T., and Zeilfelder, F. GPU-accelerated 2D point cloud visualization using smooth splines for visual analytics applications. In *Spring Conference on Computer Graphics* (2008), pp. 111–125.

[6] Kalbe, T., and Zeilfelder, F. Hardware-accelerated, high-quality rendering based on trivariate splines approximating volume data. *Computer Graphics Forum, special issue Proceedings Eurographics 27*, 2 (2008), 331–340.

[7] Marinc, A., Kalbe, T., Rhein, M., and Goesele, M. Interactive isosurfaces with quadratic C^1-splines on truncated octahedral partitions. In *Conference on Visualization and Data Analysis (VDA)* (2011).

[8] Rhein, M., and Kalbe, T. Quasi-interpolation by quadratic C^1-splines on truncated octahedral partitions. *Computer Aided Geometric Design 26*, 8 (2009), 825–841.

[9] Schwarzkopf, A., Kalbe, T., Kuijper, A., Goesele, M., and Bajaj, C. Volumetric nonlinear anisotropic diffusion on GPUs. In *Proceedings International Scale Space and Variational Methods (SSVM)* (2011).

[10] Weber, D., Kalbe, T., Stork, A., Goesele, M., and Fellner, D. Interactive deformable models with quadratic bases in Bernstein-Bézier form. In *Proceedings Computer Graphics International* (2011).

Bibliography

[AA03] Adamson A., Alexa M.: Ray tracing point set surfaces. In *Proceedings Shape Modeling International (SMI)* (2003), IEEE, p. 272.

[ABCO*01] Alexa M., Behr J., Cohen-Or D., Fleishman S., Levin D., Silva C.: Point set surfaces. In *Proceedings Visualization (VIS)* (2001), IEEE, pp. 21–28.

[ABCO*03] Alexa M., Behr J., Cohen-Or D., Fleishman S., Levin D., Silva C.: Computing and rendering point set surfaces. *Transactions on Visualization and Computer Graphics 9*, 1 (2003), 3–15.

[ABK98] Amenta N., Bern M., Kamvysselis M.: A new Voronoi-based surface reconstruction algorithm. In *Proceedings SIGGRAPH* (1998), ACM, pp. 415–421.

[ACDL00] Amenta N., Choi S., Dey T. K., Leekha N.: A simple algorithm for homeomorphic surface reconstruction. In *Proceedings Symposium on Computational Geometry (SCG)* (2000), ACM, pp. 213–222.

[ACK00] Amenta N., Choi S., Kolluri R.: The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications 19*, 2-3 (2000), 127–153.

[ACK01] Amenta N., Choi S., Kolluri R.: The power crust. In *Proceedings Symposium on Solid Modeling and Applications (SMA)* (2001), ACM, pp. 249–266.

[AdVDI03] Alliez P., de Verdière É. C., Devillers O., Isenburg M.: Isotropic surface remeshing. In *Proceedings Shape Modeling International (SMI)* (2003), IEEE, pp. 49–58.

[AK04] Amenta N., Kil Y.: Defining point-set surfaces. In *Proceedings SIGGRAPH* (2004), ACM, pp. 264–270.

[AL04] Awanou G., Lai M.-J.: Trivariate spline approximations of 3D Navier-Stokes equations. *Mathematics of Computation 74*, 250 (2004), 585–601.

[AL09] Aila T., Laine S.: Understanding the efficiency of ray traversal on GPUs. In *Proceedings of High-Performance Graphics (HPG)* (2009), ACM, pp. 145–150.

[AW87] Amanatides J., Woo A.: A fast voxel traversal algorithm for ray tracing. In *Proceedings Eurographics* (1987), pp. 3–10.

[Baj99] Bajaj C.: *Data Visualization Techniques.* John Wiley & Sons, 1999.

[BB97] Bernardini F., Bajaj C.: Sampling and reconstructing manifolds using Alpha-Shapes. In *Proceedings Canadian Conference on Computational Geometry* (1997), pp. 193–198.

[BB08] Brunner D., Brunnett G.: Fast force field approximation and its application to skeletonization of discrete 3D objects. *Computer Graphics Forum, special issue Eurographics Proceedings 27*, 2 (2008), 261–270.

[BCX95a] Bajaj C. L., Chen J., Xu G.: Modeling with C^2 quintic A-patches. In *4th SIAM Conference on Geometric Design* (1995).

[BCX95b] Bajaj C. L., Chen J., Xu G.: Modeling with cubic A-patches. *Transactions on Graphics 14*, 2 (1995), 103–133.

[BF80] Barnhill R. E., Farin G.: C^1 quintic interpolation over triangles: Two explicit representations. *International Journal for Numerical Methods in Engineering 17*, 12 (1980), 1763–1778.

[BF83] Boehm W., Farin G.: Concerning subdivision of Bézier triangles. *Computer Aided Design 15*, 5 (1983), 260–261.

[Ble90] Blelloch G. E.: *Vector models for data-parallel computing.* PhD thesis, 1990.

[BMDS02] Barthe L., Mora B., Dodgson N., Sabin M.: Triquadratic reconstruction for interactive modelling of potential fields. In *Proceedings Shape Modeling International (SMI)* (2002), IEEE, pp. 145–153.

[Bru98] Bruijns J.: Quadratic Bézier triangles as drawing primitives. In *Proceedings SIGGRAPH/Eurographics Workshop on Graphics Hardware* (1998), ACM, pp. 15–24.

[BS02] Bolz J., Schröder P.: Evaluation of subdivision surfaces on programmable graphics hardware, 2002.

[BS05] Boubekeur T., Schlick C.: Generic mesh refinement on GPU. In *Graphics Hardware* (2005), pp. 99–104.

[BW97] Bloomenthal J., Wyvill B. (Eds.): *Introduction to Implicit Surfaces.* Morgan Kaufmann, 1997.

[BW01] Brodlie K., Wood J.: Recent advances in volume visualization. *Computer Graphics Forum 20*, 2 (2001), 125–148.

[CH90] Chui C. K., He T. X.: Bivariate C^1 quadratic finite elements and vertex splines. *Mathematics of Computation 54*, 189 (1990), 169–187.

[Chu89] Chui C.: *Multivariate Splines*. CBMS 54, SIAM, 1989.

[CJ05] Chui C. K., Jiang Q. T.: Refinable bivariate quartic C^1-splines for multilevel data representation and surface display. *Mathematics of Computation 74*, 251 (2005), 1369–1390.

[CKY00] Chen M., Kaufman A., Yagel R.: *Volume Graphics*. Springer, 2000.

[CL96b] Curless B., Levoy M.: A volumetric method for building complex models from range images. In *Proceedings SIGGRAPH* (1996), ACM, pp. 303–312.

[CRZP04] Chen W., Ren L., Zwicker M., Pfister H.: Hardware-accelerated adaptive ewa volume splatting. In *Proceedings Visualization (VIS)* (2004), IEEE, pp. 67–74.

[CS94] Cohen D., Sheffer Z.: Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer 10*, 11 (1994), 27–38.

[Cse05] Csebfalvi B.: Prefiltered Gaussian reconstruction for high-quality rendering of volumetric data sampled on a body-centered cubic grid. In *Proceedings Visualization (VIS)* (2005), IEEE, pp. 311–318.

[CT65] Clough R., Tocher J.: Finite element stiffness matrices for the analysis of plate bending. In *Proceedings Conference on Matrix Methods in Structural Mechanics* (1965), pp. 515–545.

[CXZ05] Crawfis R., Xue D., Zhang C.: Volume rendering using splatting. In *The Visualization Handbook* (2005), Hansen C. D., Johnson C. R., (Eds.), Elsevier, pp. 175–188.

[dB87] de Boor C.: B-form basics. In *Geometric Modelling* (1987), Farin G., (Ed.), SIAM, pp. 131–148.

[dB93] de Boor C.: On evaluation of box splines. *Numerical Algorithms 5*, 1 (1993), 5–23.

[dBF73] de Boor C., Fix G. J.: Spline approximants by quasi-interpolants. *Journal of Approximation Theory 8* (1973), 19–45.

[DG01] Dey T., Giesen J.: Detecting undersampling in surface reconstruction. In *Proceedings Symposium on Computational Geometry (SCG)* (2001), ACM, pp. 257–263.

[DG04] Dey T., Goswami S.: Provable surface reconstruction from noisy samples. In *Proceedings Symposium on Computational Geometry (SCG)* (2004), ACM, pp. 330–339.

[DGH01] Dey T., Giesen J., Hudson J.: Delaunay based shape reconstruction from large data. In *Proceedings Symposium on parallel and large-data visualization and graphics (PVG)* (2001), IEEE, pp. 19–27.

[DNZ01] Davydov O., Nürnberger G., Zeilfelder F.: Bivariate spline interpolation with optimal approximation order. *Constructive Approximation 17* (2001), 181 – 208.

[DZ04] Davydov O., Zeilfelder F.: Scattered data fitting by direct extension of local polynomials to bivariate splines. *Advances in Computational Mathematics 21*, 3-4 (2004), 223–271.

[DZTS06] Dyken C., Ziegler G., Theobalt C., Seidel H.-P.: *HistoPyramids in iso-surface extraction.* Tech. rep., MPI Saarbrücken, 2006.

[EHKRS06] Engel K., Hadwiger M., Kniss J., Rezk-Salama C.: *Real-Time Volume Graphics.* A.K. Peters, 2006.

[EI07] Es A., İşler V.: Accelerated regular grid traversals using extended anisotropic chessboard distance fields on a parallel stream processor. *Journal of Parallel Distributed Computing 67*, 11 (2007), 1201–1217.

[EM92] Edelsbrunner H., Mücke E.: Three-dimensional Alpha Shapes. In *Proceedings Workshop on Volume Visualization* (1992), ACM, pp. 75–82.

[EM06] Entezari A., Möller T.: Extensions of the Zwart-Powell box spline for volumetric data reconstruction on the Cartesian lattice. *Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1337–1344.

[Eve01] Everitt C.: *Interactive order-independent transparency.* Tech. rep., NVIDIA Corp., 2001.

[FAKG10] Fuhrmann S., Ackermann J., Kalbe T., Goesele M.: Direct resampling for isotropic surface remeshing. In *Proc. Vision, Modeling and Visualization (VMV)* (2010).

[Far86] Farin G.: Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design (CAGD) 3*, 2 (1986), 83–127.

[Far02] Farin G.: *Curves and Surfaces for CAGD, 5th Edition.* Morgan-Kaufmann, 2002.

[FEVM10] Finkbeiner B., Entezari A., Ville D. V. D., Möller T.: Efficient volume rendering on the body centered cubic lattice using box splines. *Computers and Graphics 16* (2010), –.

[Fra79] Franke R.: *A critical comparison of some methods for interpolation of scattered data.* Tech. Rep. NPS-53-79-003, Naval Postgraduate School Tech.Rep., 1979.

[FZ09] Fu Y., Zhou B.: Direct sampling on surfaces for high quality remeshing. *Computer Aided Geometric Design (CAGD) 26*, 6 (2009), 711–723.

[Har98] Hartmann E.: A marching method for the triangulation of surfaces. *The Visual Computer 14*, 3 (1998), 95–108.

[HBRZ09] Hering-Bertram M., Reis G., Zeilfelder F.: Adaptive quasi-interpolating quartic splines. *Computing 86* (8 2009), 89–100.

[HDD*92] Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W.: Surface reconstruction from unorganized points. In *Proceedings SIGGRAPH* (1992), ACM, pp. 71–78.

[HE95] Herbison-Evans D.: Solving quartics and cubics for graphics. In *Graphics Gems V*, Paeth A. W., (Ed.). Morgan Kaufmann, 1995, pp. 3–15.

[HL93] Hoschek J., Lasser D.: *Fundamentals of Computer Aided Geometric Design*. A.K. Peters, 1993.

[HLRSR09] Hadwiger M., Ljung P., Rezk-Salama C., Ropinski T.: GPU-based volume ray-casting with advanced illumination. In *Annex Proceedings Eurographics* (2009), pp. 39–212.

[HNR*04] Hangelbroek T., Nürnberger G., Rössl C., Seidel H.-P., Zeilfelder F.: Dimension of C^1 splines on type-6 tetrahedral partitions. *Journal of Approximation Theory 131* (2004), 157–184.

[HNSZ08] Hecklin G., Nürnberger G., Schumaker L. L., Zeilfelder F.: A local Lagrange interpolation method based on C^1 cubic splines on Freudenthal partitions. *Mathematics of Computation 77* (2008), 1017 – 1036.

[HNSZ09] Hecklin G., Nürnberger G., Schumaker L. L., Zeilfelder F.: Local Lagrange interpolation with cubic C^1 splines on tetrahedral partitions. *Journal of Approximation Theory 160*, 1-2 (2009), 89 – 102.

[HNZ06] Hecklin G., Nürnberger G., Zeilfelder F.: Structural analysis of C^1-spline spaces on Freudenthal partitions. *Journal on Mathematical Analysis 38*, 2 (2006), 347 – 367.

[HS05] Hadwiger M., Sigg C.: Fast third-order texture filtering. In *GPU Gems II*, Pharr M., (Ed.). Addison-Wesley, 2005, pp. 313–329.

[HSO07] Harris M., Sengupta S., Owens J. D.: Parallel prefix sum (scan) with CUDA. In *GPU Gems 3*, Nguyen H., (Ed.). Addison-Wesley, 2007, pp. 851–876.

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

[HZDS01] Haber J., Zeilfelder F., Davydov O., Seidel H.: Smooth approximation and rendering of large scattered data sets. In *Proceedings Visualization (VIS)* (2001), IEEE, pp. 341–348.

[JBS06] Jones M., Baerentzen A., Sramek M.: 3D distance fields: a survey on techniques and applications. *Transactions on Visualization and Computer Graphics 12*, 4 (2006), 581–599.

[KBH06] Kazhdan M., Bolitho M., Hoppe H.: Poisson surface reconstruction. In *Proceedings Eurographics/SIGGRAPH Symposium on Geometry Processing (SGP)* (2006), ACM, pp. 61–70.

[KBSS01] Kobbelt L. P., Botsch M., Schwanecke U., Seidel H.-P.: Feature sensitive surface extraction from volume data. In *Proceedings SIGGRAPH* (2001), ACM, pp. 57–66.

[KFU*09] Kalbe T., Fuhrmann S., Uhrig S., Zeilfelder F., Kuijper A.: A new projection method for point set surfaces. In *Annex Proceedings Eurographics* (2009), pp. 77–80.

[KKG09] Kalbe T., Koch T., Goesele M.: High-quality rendering of varying isosurfaces with cubic trivariate C^1-continuous splines. In *Proceedings International Symposium on Visual Computing (ISVC)* (2009), pp. 596–607.

[KM05] Kaufman A., Mueller K.: Overview of Volume Rendering. In *The Visualization Handbook* (2005), Academic Press.

[Kol05] Kolluri R.: Provably good moving least squares. In *SIGGRAPH Course Notes* (2005), ACM, p. 213.

[Koo07] Koonce R.: Deferred shading in tabula rasa. In *GPU Gems 3*, Nguyen H., (Ed.). Addison-Wesley, 2007.

[KOR08] Kloetzli J., Olano M., Rheingans P.: Interactive volume isosurface rendering using bt volumes. In *Proceedings Symposium on Interactive 3D graphics and games (I3D)* (2008), ACM, pp. 45–52.

[KS01] Karkanis T., Stewart A. J.: High quality, curvature dependent triangulation of implicit surfaces. *Computer Graphics and Applications 21*, 2 (2001), 60–69.

[KTSZ08] Kalbe T., Tekušová T., Schreck T., Zeilfelder F.: GPU-accelerated 2D point cloud visualization using smooth splines for visual analytics applications. In *Proceedings Spring Conference on Computer Graphics (SCCG)* (2008), pp. 111–125.

[KW03] Krüger J., Westermann R.: Acceleration techniques for GPU-based volume rendering. In *Proceedings Visualization (VIS)* (2003), IEEE, pp. 287–292.

[KW05] Kipfer P., Westermann R.: GPU construction and transparent rendering of iso-surfaces. In *Proceedings Vision, Modeling, and Visualization (VMV)* (2005), Greiner G., Hornegger J., Niemann H., Stamminger M., (Eds.), pp. 241–248.

[KZ08] Kalbe T., Zeilfelder F.: Hardware-accelerated, high-quality rendering based on trivariate splines approximating volume data. *Computer Graphics Forum, special issue Proceedings Eurographics 27*, 2 (2008), 331–340.

[LB06] Loop C., Blinn J.: Real-time GPU rendering of piecewise algebraic surfaces. *Transactions on Graphics 25*, 3 (2006), 664–670.

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

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

[Lev90] Levoy M.: Efficient ray tracing of volume data. *Transactions on Graphics 9*, 3 (1990), 245–261.

[Lev98] Levin D.: The approximation power of moving least-squares. *Mathematics of Computation 67*, 224 (1998), 1517–1531.

[LGM*08] Ledergerber C., Guennebaud G., Meyer M., Bacher M., Pfister H.: Volume MLS ray casting. *Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1372 – 1379.

[LHJ99] LaMar E., Hamann B., Joy K.: High-quality rendering of smooth isosurfaces. *Journal of Visualization and Computer Animation 10* (1999), 79–90.

[LHK*04] Luebke D., Harris M., Krüger J., Purcell T., Govindaraju N., Buck I., Woolley C., Lefohn A.: GPGPU: general purpose computation on graphics hardware. In *SIGGRAPH Course Notes* (2004), ACM, p. 33.

[LHLW09] Liu F., Huang M.-C., Liu X.-H., Wu E.-H.: Efficient depth peeling via bucket sort. In *Proceedings of High-Performance Graphics (HPG)* (2009), ACM, pp. 51–57.

[LL94] Lacroute P., Levoy M.: Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics 28*, Annual Conference Series (1994), 451–458.

[LM04] Lai M.-J., Méhauté A. L.: A new kind of trivariate C^1 spline. *Advances in Computational Mathematics (special issue: multivariate splines) 21* (2004), 273–292.

[Loo87] Loop C.: *Smooth subdivision surfaces based on triangles.* PhD thesis, 1987.

[LPC*00] Levoy M., Pulli K., Curless B., Rusinkiewicz S., Koller D., Pereira L., Ginzton M., Anderson S., Davis J., Ginsberg J., Shade J., Fulk D.: The Digital Michelangelo Project: 3D scanning of large statues. In *Proceedings SIGGRAPH* (2000), ACM, pp. 131–144.

[LS07] Lai M.-J., Schumaker L.: *Spline Functions on Triangulations.* Cambridge Universiy Press, 2007.

[LS08] Loop C., Schaeffer S.: Approximating Catmull-Clark subdivision surfaces with bicubic patches. *Transactions on Graphics 27*, 1 (2008).

[LW01] Lai M. J., Wenston P.: Trivariate C^1 cubic splines for numerical solution of biharmonic equations. In *Trends in Approximation Theory* (2001), pp. 224–233.

[MJC00] Mora B., Jessel J.-P., Caubet R.: Accelerating volume rendering with quantized voxels. In *Proceedings Symposium on Volume Visualization* (2000), IEEE, pp. 63–70.

[MKRG11] Marinc A., Kalbe T., Rhein M., Goesele M.: Interactive isosurfaces with quadratic C^1-splines on truncated octahedral partitions. In *Conference on Visualization and Data Analysis (VDA)* (2011).

[ML94] Marschner S., Lobb R.: An evaluation of reconstruction filters for volume rendering. In *Proceedings Visualization (VIS)* (1994), IEEE, pp. 100–107.

[MMC99] Mueller K., Möller T., Crawfis R.: Splatting without the blur. In *Proceedings Visualization (VIS)* (1999), IEEE.

[MRH08] Mensmann J., Ropinski T., Hinrichs K.: Accelerating volume raycasting using occlusion frustum. *IEEE/Eurographics International Symposium on Volume and Point-Based Graphics* (2008), 147–154.

[MS07] Manni C., Sablonnière P.: Quadratic spline quasi-interpolants on Powell-Sabin partitions. *Advances in Computational Mathematics 26*, 1-3 (2007), 283–304.

[Nie00] Nielson G.: Volume modelling. In *Volume Graphics* (2000), Chen M., Kaufman A., R.Yagel, (Eds.), Springer, pp. 29–50.

[Nie04] Nielson G.: Dual marching cubes. In *Proceedings Visualization (VIS)* (2004), IEEE, pp. 489–496.

[NM05] Neophytou N., Mueller K.: GPU accelerated image aligned splatting. In *Fourth International Workshop on Volume Graphics* (2005), pp. 197–242.

[NRSZ05] Nürnberger G., Rössl C., Seidel H.-P., Zeilfelder F.: Quasi-interpolation by quadratic piecewise polynomials in three variables. *Computer Aided Geometric Design (CAGD) 22* (2005), 221–249.

[NRSZ06] Nürnberger G., Rayevskaya V., Schumaker L., Zeilfelder F.: Local Lagrange interpolation with bivariate splines of arbitrary smoothness. *Constructive Approximation 23* (2006), 33 – 59.

[NRZ07] Nürnberger G., Rössl C., Zeilfelder F.: High-quality rendering of iso-surfaces extracted from quadratic super splines. In *Proceedings Curves and Surfaces, Avignon* (2007).

[NSZ05] Nürnberger G., Schumaker L., Zeilfelder F.: Two Lagrange interpolation methods based on C^1 splines on tetrahedral partitions. In *Approximation Theory XI: Gatlinburg* (2005), Chui C. K., (Ed.), pp. 101–118.

[NVI08a] NVIDIA: *CUDA Compute Unified Device Architecture.* Tech. rep., 2008.

[NVI08b] NVIDIA: *GPU Programming Guide. GeForce 8 and 9 Series.* Tech. rep., 2008.

[NVI09a] NVIDIA: *CUDA C Programming Best Practices Guide.* Tech. rep., 2009.

[NVI09b] NVIDIA: *CUDA Programming Guide 2.3.1.* Tech. rep., 2009.

[NY06] Newman S., Yi H.: A survey of the marching cubes algorithm. *Computers & Graphics 30* (2006), 845–879.

[NZ00] Nürnberger G., Zeilfelder F.: Developments in bivariate spline interpolation. *Journal of Computational and Applied Mathematics 121* (2000), 125–152.

[PBP02] Prautzsch H., Boehm W., Paluszny M.: *Bézier and B-Spline techniques.* Springer, 2002.

[PS77] Powell M. J. D., Sabin M. A.: Piecewise quadratic approximations on triangles. *Transactions on Mathematical Software (TOMS) 3*, 4 (1977), 316–325.

[PSL*98] Parker S., Shirley P., Livnat Y., Hansen C., Sloan P.-P.: Interactive ray tracing for isosurface rendering. In *Proceedings Visualization (VIS)* (1998), IEEE, pp. 233–238.

[Ram87] Ramshaw L.: *Blossoming: A connect-the-dots approach to splines.* Tech. rep., Digital Systems Research Center, 1987.

[Rei05] Reis G.: Hardware based Bézier patch renderer. In *Proceedings of IASTED Visualization, Imaging, and Image Processing (VIIP)* (2005), pp. 622–627.

[RK09] Rhein M., Kalbe T.: Quasi-interpolation by quadratic C^1-splines on truncated octahedral partitions. *Computer Aided Geometric Design (CAGD) 26*, 8 (2009), 825–841.

[RZHB*08] Reis G., Zeilfelder F., Hering-Bertram M., Farin G., Hagen H.: High-quality rendering of quartic spline surfaces on the GPU. *Transactions on Visualization and Computer Graphics 14*, 5 (2008), 1126–1139.

[RZNS03] Rössl C., Zeilfelder F., Nürnberger G., Seidel H.-P.: Visualization of volume data with quadratic super splines. In *Proceedings Visualization (VIS)* (2003), IEEE, pp. 393–400.

[RZNS04a] Rössl C., Zeilfelder F., Nürnberger G., Seidel H.-P.: Reconstruction of volume data with quadratic super splines. *Transactions on Visualization and Computer Graphics 4*, 10 (2004), 397–409.

[RZNS04b] Rössl C., Zeilfelder F., Nürnberger G., Seidel H.-P.: Spline approximation of general volumetric data. *Symposium on Solid and Physical Modeling* (2004), 74–82.

[Sch89] Schumaker L. L.: On super splines and finite elements. *Journal on Numerical Analysis 26*, 4 (1989), 997–1005.

[Sch90] Schwarze J.: Cubic and quartic roots. In *Graphics Gems*, Glassner A., (Ed.). Academic Press, 1990, pp. 404–407.

[SD07] Seland J., Dokken T.: Real-time algebraic surface visualization. In *Geometric Modelling, Numerical Simulation, and Optimization* (2007), Springer, pp. 163–183.

[Sei89] Seidel H.-P.: A general subdivision theorem for Bézier triangles. In *Mathematical Methods in Computer Aided Design* (1989), Lyche T., Schumaker L., (Eds.), Academic Press, pp. 573–582.

[Sei93] Seidel H.-P.: An introduction to polar forms. *Computer Graphics and Applications 13*, 1 (1993), 38–46.

[SFS05] Scheidegger C., Fleishman S., Silva C.: Triangulating point-set surfaces with bounded error. In *Proceedings Eurographics/ACM Symposium on Geometry Processing* (2005), Desbrun M., Pottman H., (Eds.), pp. 63–72.

[SG03] Surazhsky V., Gotsman C.: Explicit surface remeshing. In *Proceedings Eurographics/SIGGRAPH Symposium on Geometry Processing (SGP)* (2003), Eurographics Association, pp. 20–30.

[SGS06] Stoll C., Gumhold S., Seidel H.-P.: Incremental raycasting of piecewise quadratic surfaces on the GPU. In *Proceedings Symposium on Interactive Raytracing* (2006), IEEE, pp. 141–150.

[SJP05] Shiue L.-J., Jones I., Peters J.: A realtime GPU subdivision kernel. In *Proceedings SIGGRAPH* (2005), ACM, pp. 1010–1015.

[SOM04] Sud A., Otaduy M., Manocha D.: Difi: fast 3D distance field computation using graphics hardware. *Computer Graphics Forum 23*, 3 (2004).

[Sor06] Sorokina T.: On the n-dimensional Clough-Tocher interpolant, 2006.

[SS04] Schumaker L., Sorokina T.: C^1 quintic splines on type-4 tetrahedral partitions. *Advances in Computational Mathematics 21* (2004), 412–444.

[SS06] Stelldinger P., Strand R.: Topology preserving digitization with FCC and BCC grids. In *Lecture Notes in Computer Science: Combinational Image Analysis* (2006), Springer Heidelberg, pp. 226–240.

[SSFS07] Schreiner J., Scheidegger C., Fleishman S., Silva C.: Afront. http://afront.sourceforge.net/, 2007.

[SSS06a] Schreiner J., Scheidegger C., Silva C.: High quality extraction of isosurfaces from regular and irregular grids. *Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1205–1212.

[SSS06b] Schwarz M., Staginski M., Stamminger M.: GPU-based rendering of PN triangle meshes with adaptive tesselation. In *Proceedings Vision, Modeling, and Visualization (VMV)* (2006), pp. 161 – 168.

[SSW09] Schumaker L., Sorokina T., Worsey A.: A C^1 quadratic trivariate macro-element space defined over arbitrary tetrahedral partitions. *Journal of Approximation Theory 158*, 1 (2009), 126–142.

[ST90] Shirley P., Tuchman A.: A polygonal approximation to direct scalar volume rendering. *SIGGRAPH Computer Graphics 24*, 5 (1990), 63–70.

[SW04] Schaefer S., Warren J.: Dual marching cubes: Primal contouring of dual grids. In *Proceedings Computer Graphics and Applications* (2004), IEEE, pp. 70–76.

[SW08] Sorokina T., Worsey A.: A multivariate Powell-Sabin interpolant. *Advances in Computational Mathematics 29*, 1 (2008), 71–89.

[SWBG06] Sigg C., Weyrich T., Botsch M., Gross M.: GPU-based ray-casting of quadratic surfaces. In *Proceedings Eurographics Symposium on Point-Based Graphics* (2006), pp. 59–65.

[SZ05] Sorokina T., Zeilfelder F.: Optimal quasi-interpolation by quadratic C^1 splines on four-directional meshes. In *Approximation Theory XI: Gatlinburg* (2005), Chui C. K., Neamtu M., Schumaker L. L., (Eds.), pp. 423–438.

[SZ07b] Sorokina T., Zeilfelder F.: Local quasi-interpolation by cubic C^1 splines on type-6 tetrahedral partitions. *IMJ Numerical Analysis 27* (2007), 74–101.

[SZH*05] Schlosser G., Zeilfelder F., Hesser J., Rössl C., Nürnberger G., Männer R., Seidel H.-P.: Fast visualization by shear-warp on quadratic super-spline models using wavelet data decompositions. IEEE, pp. 45–55.

[The02] Theisel H.: Exact isosurfaces for marching cubes. *Computer Graphics Forum 21*, 1 (2002), 19–31.

[TL93] Totskua T., Levoy M.: Frequency domain volume rendering. In *Proceedings SIGGRAPH* (1993), ACM, pp. 271–278.

[TL94a] Turk G., Levoy M.: Zippered polygon meshes from range images. In *Proceedings SIGGRAPH* (1994), ACM, pp. 311–318.

[TMHG01] Theussl T., Möller T., Hladůvka J., Gröller M. E.: *Reconstruction issues in volume visualization.* Tech. Rep. TR-186-2-01-14, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2001.

[TSD07] Tatarchuk N., Shopf J., DeCoro C.: Real-time isosurface extraction using the GPU programmable geometry pipeline. In *SIGGRAPH Course Notes* (2007), ACM, pp. 122–137.

[TU01] Thévenaz P., Unser M.: High-quality isosurface rendering with exact gradients. In *Proceedings Visualization (VIS)* (2001), IEEE, pp. 854–857.

[Uhr08] Uhrig S.: Cloudmesh. http://sourceforge.net/projects/cloudmesh, 2008.

[VdFG98] Velho L., de Figueiredo L. H., Gomes J. A.: *Implicit Objects in Computer Graphics.* Springer New York, 1998.

[vK00] Šrámek M., Kaufman A.: Fast ray-tracing of rectilinear volume data using distance transforms. *Transactions on Visualization and Computer Graphics 6*, 3 (2000), 236–252.

[VKG04] Viola I., Kanitsar A., Gröller M.: GPU-based frequency domain volume rendering. In *Proceedings Spring Conference on Computer Graphics (SCCG)* (2004), pp. 49–58.

[VPBM01] Vlachos A., Peters J., Boyd C., Mitchell J. L.: Curved PN triangles. In *Proceedings Symposium on Interactive 3D graphics (I3D)* (2001), ACM, pp. 159–166.

[Ž70] Ženišek A.: Interpolation polynomials on the triangle. *Numerische Mathematik 15*, 4 (1970), 283–296.

[Wes89] Westover L.: Interactive volume rendering. *Chapel Hill Volume Visualization Workshop* (1989), 9–16.

[Wes90] Westover L.: Footprint evaluation for volume rendering. *Proceedings SIGGRAPH 24* (1990), 367–376.

[WF87] Worsey A. J., Farin G.: An n-dimensional Clough-Tocher interpolant. *Constructive Approximation 3*, 1 (1987), 99–110.

[WKG*11] Weber D., Kalbe T., Goesele M., Stork A., Fellner D.: Interactive deformable models with quadratic bases in Bernstein-Bézier form. In *Proceedings Computer Graphics International* (2011).

[WKME03] Weiler M., Kraus M., Merz M., Ertl T.: Hardware-based ray casting for tetrahedral meshes. In *Proceedings Visualization (VIS)* (2003), IEEE, pp. 333–340.

[WMK04] Wood A., McCane B., King S. A.: Ray tracing arbitrary objects on the GPU. In *Proceedings Image and Vision Computing New Zealand (IVCNZ)* (2004), pp. 327–332.

[WMW86] Wyvill G., McPhetters C., Wyvill B.: Data structures for soft objects. *The Visual Computer 2* (1986), 227–234.

[WP88] Worsey A. J., Piper B.: A trivariate Powell-Sabin interpolant. *Computer Aided Geometric Design (CAGD) 5*, 3 (1988), 177–186.

[WS01] Wald I., Slusallek P.: State of the art in interactive ray tracing. In *STAR, EUROGRAPHICS*. 2001, pp. 21–42.

[XZC05] Xue D., Zhang C., Crawfis R.: iSBVR: Isosurface-aided hardware acceleration techniques for slice-based volume rendering. In *Volume Graphics* (2005), Kaufman A. E., Mueller K., Gröller E., Fellner D. W., Müller T., Spencer S. N., (Eds.), Eurographics Association, pp. 207–215.

[Zei02] Zeilfelder F.: Scattered data fitting with bivariate splines. In *Tutorials on Multiresolution in Geometric Modeling* (2002), Iske A., Quak E., Floater M. S., (Eds.), Springer, pp. 243–283.

[ZPvBG02] Zwicker M., Pfister H., van Baar J., Gross M.: Ewa splatting. *Transactions on Visualization and Computer Graphics 8*, 3 (2002), 223–238.

[ZXB07] Zhao W., Xu G., Bajaj C.: An algebraic spline model of molecular surfaces. In *Proceedings Symposium on Solid and Physical Modeling (SPM)* (2007), pp. 297–302.