Towards Meshless Volume Visualization

Von der Fakultät Informatik, Elektrotechnik und Informations-
technik der Universität Stuttgart zur Erlangung der Würde
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Eduardo Jose Tejada-Gamero

aus Arequipa

Hauptberichter:	Prof. Dr. T. Ertl
Mitberichter:	Prof. Dr. D. Weiskopf
	Prof. Dr. L. G. Nonato
Tag der mündlichen Prüfung:	26. März 2008

Institut für Visualisierung und Interaktive Systeme
der Universität Stuttgart

2008

Berichte aus der Informatik

Eduardo Jose Tejada-Gamero

Towards Meshless Volume Visualization

Gedruckt mit Unterstützung des Deutschen Akademischen Austauschdienstes

D93 (Diss. Universität Stuttgart)

Shaker Verlag
Aachen 2008

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: Stuttgart, Univ., Diss., 2008

ISBN 978-3-8322-7389-7
ISSN 0945-0807

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

A Paola y Andrés,
quienes son la luz en mi vida.

CONTENTS

List of Abbreviations and Acronyms **1**

Abstract and Chapter Summaries **3**

Zusammenfassung und Kapitelzusammenfassungen **9**

1 Introduction **15**

1.1 Goals of This Thesis 16
1.2 Outline of This Thesis 17
1.3 Acknowledgments 18

2 Interactive Visualization **21**

2.1 Visualization Pipeline 21
2.2 Surface Visualization 23
2.2.1 Surface data 23
2.2.2 Surface reconstruction 23
2.2.3 Surface rendering 25
2.3 Volume Visualization 27
2.3.1 Volume data 27
2.3.2 Volume data reconstruction 29
2.3.3 Volume rendering 35
2.4 Visualization and Graphics Processing Units 41
2.4.1 The rendering pipeline 41
2.4.2 General-purpose GPU programming 43

3 Meshless Approximation Methods **45**

3.1 Radial Basis Functions 45
3.2 Moving Least-Squares 47
3.3 Orthogonal Polynomials in Moving Least-Squares 47
3.3.1 Indexing orthogonal polynomials 48
3.3.2 Constructing orthogonal polynomials 49
3.3.3 Avoiding repetitive computations 50
3.4 Approximate Approximation 51
3.4.1 Approximate moving least-squares approximation 51
3.4.2 Connecting RBF and Iterated AMLS 52

4 Meshless Surfaces from Point Clouds 55
4.1 Meshless Surface Approximation . . . 56
4.2 Curvature-driven Projection Operator . . . 62
4.2.1 Principal directions and curvatures . . . 64
4.2.2 Projection and rendering procedures . . . 66
4.3 Approximate MLS Surfaces . . . 71
4.3.1 Iterated AMLS implicits . . . 74
4.3.2 Introducing sharp edges . . . 79
4.4 Adaptive Partition of Unity Implicits . . . 82
4.4.1 Multi-level partition of unity implicits . . . 84
4.4.2 The J_A^1 triangulation . . . 85
4.4.3 Robust adaptive partition of unity implicits . . . 87
4.4.4 Extensions to the method . . . 91
4.5 GPU-based Rendering of Meshless Surfaces . . . 96
4.5.1 Rendering surfaces based on projection operators . . . 97
4.5.2 Rendering implicit surfaces . . . 101

5 Meshless Surfaces from Volumes 105
5.1 Meshless Surface Extraction from Volume Data . . . 105
5.2 Moving Least-squares Iso-surfaces . . . 108
5.2.1 Computing MLS surfaces from volumetric data . . . 109
5.2.2 Hardware-accelerated MLS Iso-surfaces and HG-surfaces 110
5.3 Point-based Stream Surfaces . . . 114
5.3.1 Streamlines and path-lines generation . . . 115
5.3.2 Point-based surface rendering . . . 120
5.3.3 LIC on the point-based surface . . . 122

6 Meshless Volume Visualization 127
6.1 Meshless Methods for Volume Visualization . . . 129
6.2 Moving Least-Squares Volume Visualization . . . 131
6.2.1 Detail-preserving volume data approximation . . . 132
6.2.2 Matrix-free detail-preserving volume data approximation . 134
6.3 Approximate MLS Volume Visualization . . . 138
6.3.1 Ellipsoidal weight functions revisited . . . 139
6.3.2 Anisotropic iterated approximate moving least-squares . . 140
6.3.3 Gradient estimation . . . 142
6.3.4 GPU-based rendering . . . 143
6.4 Moving Least-Squares Volume Deformation . . . 147
6.4.1 Affine, similarity and rigid deformations . . . 148
6.4.2 Nonlinear polynomial deformation . . . 155
6.4.3 GPU-based MLS displacement map computation . . . 156

6.4.4 Other approaches for moving least-squares deformation . 161
6.4.5 Comparison with physically-based mesh deformation . . . 164

7 Meshless Methods in Visualization 171

Color Plates 177

Bibliography 189

List of Abbreviations and Acronyms

AMLS	Approximate Moving Least-Squares	i.e.	id est
bit	binary digit	LIC	Line Integral Convolution
bspw.	beispielsweise	LS	Least-Squares
CG	Conjugate Gradient	MB	megabyte
CPU	Central Processing Unit	MLS	Moving Least-Squares
CUDA	Compute Unified Device Architexture	MRI	Magnetic Resonance Imaging
CT	Computer Tomography	PC	Personal Computer
CTM	Close To Metal	pixel	picture element
Dr. rer. nat.	Doctor rerum naturalium	Prof. Dr.	Professor Doctor
EBF	Elliptical Basis Functions	PU	Partition of Unity
e.g.	exempli gratia	RAM	Random Access Memory
et al.	et alii, et aliae, et alia	RBF	Radial Basis Functions
etc.	et cetera	RGB	red, green, blue
fps	frames per second	RGBA	red, green, blue, alpha
GB	gigabyte	s	second
GHz	gigahertz	SIMD	Single Instruction, Multiple Data
GPGPU	general computations on the GPU	surfel	surface element
GPU	Graphics Processing Unit	SVD	Singular Value Decomposition
HG	high-gradient	texel	texture element
IAMLS	Iterated Approximate Moving Least-Squares	voxel	volume element
		WLS	Weighted Least-Squares

Abstract and Chapter Summaries

Abstract

In this thesis, novel meshless methods for surface and volume data reconstruction and rendering are proposed. Surface reconstruction from unorganized point sets is first addressed with projection operators. Specifically, a curvature-driven projection operator is presented which defines an approximate surface for a given point cloud based on a diffusion equation and on curvature estimation for point sets. Implicit formulations for surface approximation are also addressed. An implicit surface definition based on approximate moving least-squares approximation is introduced, which is able to provide high-order local approximations to the surface without requiring to solve systems of equations. Bilateral filters are introduced into this surface definition in order to better represent sharp features by robustly estimating normal vectors. An adaptive implicit formulation based on partition of unity and orthogonal polynomials is also proposed. This formulation addresses approximation and robustness issues presented by previous work on partition of unity implicits. To accelerate the rendering of these surface definitions, hardware-accelerated ray-casting of implicit surfaces and surfaces defined by projection operators is also discussed.

The results obtained for surface approximation are then applied to volume data in order to extract surfaces that represent some feature in the volume. Regarding scalar data, a moving least-squares surface definition is proposed which is able to approximate iso-surfaces and surfaces located are regions with high gradient magnitude. The rendering of such surfaces is performed on graphics hardware to accelerate the computations. Visualization of vector fields is also addressed, specifically the interactive computation and rendering of streamsurfaces and of the novel path-surfaces. To that end, a hardware-accelerated streamlines and path-lines generation process is presented, which is able to produce a quasi-regular sampling of the surface. This allows the use of known point-based surface rendering algorithms to interactively visualize the streamsurface or path-surface.

Volume visualization is then addressed using meshless methods. These visualization methods are based on a meshless volume model extracted from the data. This model is obtained using the moving least-squares approximation method. In order to preserve details in the reconstruction of the volumetric data, bilateral filtering is used which, together with the use of orthogonal polynomials, provides a matrix-free detail-preserving reconstruction of the volume data. To further accelerate the computation of the function reconstruction, the use of approximate approximation is also explored in this context. To that end, an anisotropic iterated approximate moving least-squares approximation of the volume data is defined, which converges to an ellipsoidal-basis-functions interpolation of the data. Finally, volume deformation by means of moving least-squares is addressed and a closed formulation for nonlinear polynomial deformations is proposed. An implementation of the set of moving least-squares deformations on hardware graphics is also presented and used to interactively compute volume deformations by means of displacement maps.

Chapter Summaries

An overview of this thesis is given in this section as chapter summaries.

Chapter 1: Introduction

This chapter introduces the topic of the thesis. The need for a reconstruction of the volume model in the context of volume visualization is used as starting point for the discussion. The problem of reconstructing the underlying function which represents the volume model using meshless methods is the goal of this work. Thus, the methods developed were based on the success of meshless techniques in solving problems from surface modeling and rendering, which techniques based on combinatorial structures have failed to solve in the past. This leads to a brief description of the research performed, initially in the context of meshless surfaces from point clouds and volumes and latter of meshless volume visualization.

Chapter 2: Interactive Visualization

In this chapter an overview of topics on interactive visualization is given. The chapter starts by describing the visualization pipeline and its stages. Then a brief discussion on surface data, modeling and rendering is given. Since the focus of the research reported in this work is on meshless methods, unorganized point sets and polygon soups are mentioned as the primary source of data for modeling and rendering algorithms based on meshless techniques.

Groundbreaking work on meshless surface reconstruction, which set the basis for the development of the area in the last decade, is then described. These approaches represent the main trends in meshless surface reconstruction that have gained the attention of the community lately; namely, surface representations based on projection operators and surface representations based on implicit formulations. The methods based on projection operators define the approximate surface as the set of static points for a certain map, while the methods based on implicit formulations define the approximate surface as the set of points belonging, usually, to the zero set of the function.

Meshless surface rendering is then addressed. Rendering methods based on points have gained popularity in the last years. Many works have been proposed and their detailed description is beyond the scope of this thesis. Nonetheless, the main ideas are mentioned with emphasis on ray-casting, specifically on the computation of the intersection of a ray with the meshless surface. In latter chapters this description is detailed further in the context of the techniques presented.

After addressing surface visualization in the context of the visualization pipeline, the same approach is taken for volume visualization. Starting with the volume data types usually found in various applications areas, the different grid types that must be handled by volume visualization methods are described. The sources of the volume data are many and different, which in turn translates into a large variation in the nature of the volume data, including the grid type, which is of particular interest to the techniques presented latter in the thesis. This is due to fact that part of the research described in this work is focused on providing a means to reconstruct volume data stored in meshes of arbitrary topology and geometry.

After addressing reconstruction methods, volume rendering methods are then described, specifically direct volume rendering. Therefore, the derivation of the rendering integral for the emission-absorption model is given. Finally, the chapter ends with basic concepts of graphics processing units programming. The rendering pipeline implemented by most commodity graphics hardware is described. Since graphics processing units were used to a large extent in the approaches described in this work in order to accelerate the computations, the concept of general purpose programing using graphics processing units is also introduced.

Chapter 3: Meshless Approximation Methods

In this chapter, an overview of the meshless approximation methods used throughout the thesis is given. The scattered data approximation and interpolation problems are defined followed by a description of radial basis functions, where the concept of radial basis functions interpolation is introduced as well as the most widely known radial functions. Then, the polynomial moving least-squares approximation is approached with a simple and general definition.

With this basis set, orthogonal polynomials in the context of moving least-squares are addressed. For this, the concept of orthogonality of a polynomial basis for a specific inner product defined by a given weighting function is introduced. Then, an indexing that has proven to be efficient in reducing the number of operations performed to orthogonalize a polynomial basis with the Gram-Schmidt orthogonalization is described.

Lastly, approximate moving least-squares approximation is described as an efficient and matrix-free approach to approximate scattered data. This method produces an approximation to the solution of the moving least-squares method and is based on specific generating functions by means of which solutions of different approximation orders can be obtained. Radial basis functions and approximate approximation have been recently connected through an iterative process that, starting from an approximate approximation, converges to a radial basis functions interpolation. This process is also briefly described here.

Chapter 4: Meshless Surfaces from Point Clouds

This chapter presents novel approaches for surface reconstruction from unorganized point sets. Projection operators are firstly addressed. In this context, a novel curvature-driven projection operator is proposed, which is based on the computation of a non-complete second degree polynomial defined by the principal curvatures and directions at a given point on the surface. Therefore, the robust computation of the principal curvatures and directions for a set of points is addressed. This result is then used, together with an anisotropic diffusion equation, to define the projection operator, which is included in a ray-casting engine to render the surface.

Implicit surfaces are also addressed in this chapter. Two different implicit approaches are taken to tackle the problem of surface reconstruction from point clouds. The first approach proposed is based on moving least-squares surfaces and addresses approximation and performance issues presented by them. By using approximate approximation, the reconstruction process is efficiently performed while enabling the computation of high-order approximations to the surface. Furthermore, the iterative process mentioned above to obtain a radial basis functions interpolation can be used to produce interpolating surfaces. The efficiency of the method is exploited to introduce bilateral filtering in the process in order to robustly estimate the normal vectors at the surface points. This makes it possible to better visually represent sharp edges since the changes in the normal field on the surface are more important to the perception of sharp features than the approximation to the surface itself.

The second approach taken to reconstruct surfaces from point clouds is based on partition of unity implicits. Firstly, a review of the original method is given. This method presents robustness issues, which are addressed in this chapter. Furthermore, the method proposed here is twofold adaptive in that the space is adaptively partitioned to fit the details on the surface and the degree of the polynomial approximation is set adaptively to better approximate the surface. This can be done by means of orthogonal polynomials. This, however, introduces robustness problems, which are addressed by developing coverage domain criteria that guide the approximation process. Moreover, an interactive tool that enables the user to edit the surface is also presented, which allows the user to correct errors in the approximate surface or to improve the quality of the approximation. Lastly, the mesh resulting from the triangulation performed on the data structure used, the J_A^1 , presents triangles with low quality. To solve this problem, a mesh enhancement procedure based on vertex displacements is applied as a post-processing. This procedure successfully improves the quality of the triangles.

In the final section of this chapter, commodity graphics hardware is used to accelerate the rendering of surfaces defined with projection operators and with implicit functions. Specifically, a ray-tracing engine implemented on graphics hardware able to render such meshless surfaces is described.

Chapter 5: Meshless Surfaces from Volumes

Meshless methods can be exploited in volume visualization as well. Thus, in this chapter, techniques addressing two different visualization problems are introduced. The first problem is the extraction of surfaces representing meaningful information from volume scalar data; in this case, isosurfaces and surfaces located at regions of high gradient magnitude. This is done by defining suitable weighting functions and using them in the moving least-squares surface approximation method. Moreover, a novel combination of a weighted least-squares approach with the moving least-squares approximation, in a predictor-corrector sense, widens the domain of the definition allowing to project points far from the surface onto it. This process is implemented in the hardware-accelerated ray-casting engine and applied to Cartesian grids. To that end, the implementation is accommodated to fit the nature of the data and of the projection process.

The second problem addressed is the interactive generation of streamsurfaces and of the novel path-surfaces proposed here. The main issue in visualizing vector fields using streamsurfaces is the need for triangulating the input streamlines. This process can be slow and if the user interactively changes the seed points, this could lead to long waiting times. This problem is more critical in the case of path-surfaces which are generated from path-lines since the vector field is time-dependent. Thus, streamlines and path-lines are generated on the fly in this work using graphics hardware, which allows to interactively reconstruct them when the user modifies the position of the seeding points. Moreover, to be able to better support point-based rendering methods, the density of the streamlines and path-lines is maintained nearly constant by adaptively seeding and removing lines according to the evolution of the integration. This allows the use of splatting to render the surface which eliminates the need for triangulating it. Furthermore, line integral convolution is calculated on the surface to better depict the details of the flow.

Chapter 6: Meshless Volume Visualization

This chapter proposes novel methods for visualizing volume data stored in meshes of arbitrary topology. These methods are based on meshless approximation techniques to reconstruct the underlying function in the data. This problem is addressed using two different approaches. The first approach is based on a detail-preserving approximation of the volume data, obtained by minimizing a specific function. However, this problem is ill-conditioned and, since the formulation results in an iterative method based on moving least-squares and bilateral filtering, the performance is considerably reduced compared to other simpler, albeit unstable, meshless methods. Therefore, orthogonal polynomials are used to accelerate the computation of the approximation in each iteration while improving stability. This also allows to implement the technique on graphics hardware since no system of equations must be solved.

A less accurate approach based on approximate approximation, able to reduce computation times considerably, is also proposed. This approach uses the iterated approximate approximation method to produce a result that approximates a radial basis functions interpolation of the data. This way, beside increasing performance in comparison to both moving least-squares approximation and radial basis functions interpolation, a compromise between accuracy and robustness to noise can be achieved. However, like all meshless methods applied on anisotropic domains, the method must adapt to the anisotropy, which in the case of grids containing volume data is given by the configuration of the grid elements influencing the result at the evaluation point. Therefore, ellipsoidal weight functions are introduced into the process and a novel anisotropic iterated approximate approximation is defined. Thus, the iterative process converges to an elliptical basis functions interpolation. This method is also implemented on graphics hardware to reconstruct the function for performing ray-casting.

Lastly, volume manipulation using moving least-squares is also presented. To that end, previously developed methods for image and surface deformation based on moving least-squares are extended to volumes and nonlinear polynomial deformations are introduced. The key point of this novel nonlinear moving least-squares deformation is the closed formulation provided, which is one of the main advantages of the rigid, similarity and affine moving least-squares deformations proposed previously by other authors. Thus, the nonlinear deformation is an addition to the set of moving least-squares deformations available. This complete set of deformations was implemented in graphics hardware to accelerate the computation of displacement maps to support volumetric deformations. By redefining the deformations as backward mappings, it is possible to calculate this displacement map in a way that allows their use in commonly known hardware-accelerated volume rendering methods. The chapter finishes with a comparison of meshless deformations with physically-based deformations for tetrahedral meshes implemented on graphics hardware. Besides providing an opposing comparison case to meshless deformations, the novel description of the hardware-accelerated implementation of implicit integration methods for solving the differential equation governing the deformation is a further contribution.

Chapter 7: Meshless Methods in Visualization

The last chapter of the thesis provides guidelines for the use of the methods proposed in this work for surface and volume modeling and rendering. A discussion on the advantages and issues to be addressed in the future is also given.

Zusammenfassung und Kapitelzusammenfassungen

Zusammenfassung

Interaktive Volumenvisualisierung hat in den letzten Jahren in vielen Bereichen Anwendung gefunden. Wichtige Fortschritte wurden gemacht, welche die algorithmische Performanz sowie die Fähigkeit von Visualisierungstechniken für Volumendatenuntersuchung verbessert haben. Unabhängig von der Art der Daten und der Paradigmen der verwendeten Visualisierungstechnik muss ein Modell der Daten zur Verfügung stehen. Allerdings sind die Lösungsmethoden in den meisten Fällen nicht vorhanden und daher muss ein Volumenmodell der abgetasteten Funktion rekonstruiert werden. Für die interaktiven Visualisierungsmethoden wird meistens ein Volumenmodell gewählt ohne die ursprüngliche Lösungsmethode zu beachten. Trotz existierender Forschungen über Interpolation höherer Ordnung und Filterung von Volumendaten wird oft ein einfacheres Modell benutzt bspw. Rekonstruktion mittels linearer Interpolation.

Anderseits sind gitterlose Methoden für Oberflächenrekonstruktion populär geworden. Gitterlose Methoden haben verschiedene Vorteile, wie Skalierbarkeit auf verschiedenen Datentypen, Unabhängigkeit von expliziter Konnektivität und wenig Speicherverbrauch. Zusätzlich sind gitterlose Approximationstechniken genau und einfach zu berechnen. Theoretische Ergebnisse sowie praktische Anwendungen wurden mit Erfolg entwickelt. Zu Beginn beschäftigt sich diese Dissertation mit gitterloser Oberflächenapproximation und stellt neue Methoden in diesem Bereich vor. Die Ergebnisse werden dann auf Volumendaten angewendet, um Oberflächen zu extrahieren, welche bestimmte Eigenschaften in den Daten repräsentieren. Diese Richtung wird weiterverfolgt und Volumenvisualisierung wird dann mit gitterlosen Methoden behandelt. Diese Visualisierungsmethoden basieren auf einem gitterlosen Volumenmodell, das aus den Daten und der Konnektivitätsinformation des Gitters extrahiert wird. Das Ziel dieser Arbeit ist eine Grundlage zu bilden, um eine allgemeine Methode zu definieren, die auf verschiedenen Volumendatentypen anwendbar ist und auf Techniken basiert, die in anderen Bereichen bereits erfolgreich verwendet wurden.

Kapitelzusammenfassungen

Eine Übersicht dieser Dissertation wird in den folgenden Abschnitten als Kapitelzusammenfassungen gegeben.

Kapitel 1: Einführung

Dieses Kapitel führt in das Thema dieser Dissertation ein. Der Bedarf für eine Rekonstruktion des Volumenmodells im Kontext der Volumenvisualisierung wird als Ausgangspunkt für die Diskussion verwendet. Das Problem der Rekonstruktion der zu Grunde liegenden Funktion, welche das Volumenmodell repräsentiert, mit gitterlosen Methoden ist das Ziel dieser Arbeit. Daher basiert diese Arbeit auf dem Erfolg von gitterlosen Methoden im Rahmen der Oberflächenmodellierung und -darstellung, welche Methode, die auf kombinatorischen Strukturen basieren, bisher nicht lösen konnten. Das ist die Argumentation, die in diesem Kapitel eingeführt wird. Dies führt zu einer kurzen Beschreibung der Forschungsarbeiten, die sowohl die Rekonstruktion von gitterlosen Oberflächen aus Punktmengen und Volumina als auch die gitterloser Volumenvisualisierung behandelt.

Kapitel 2: Interaktive Visualisierung

In diesem Kapitel wird ein Überblick über interaktive Visualisierung gegeben. Das Kapitel fängt mit einer Beschreibung der Visualiserungspipeline an. Diese Pipeline wird dann für den spezifischen Fall von Oberflächen angepasst und eine kurze Diskussion über Oberflächendaten, Modellierung und Rendering wird gegeben. Da der Fokus dieser Arbeit auf gitterlose Methoden liegt, werden Punktmengen und *Polygon Soups* als elementare Datenquellen für gitterlose Modellierungs- und Renderingmethoden erwähnt.

Bezüglich der Modellierung sind Rekonstruktionsmethoden aus Punktmengen im Fokus der Diskussion. Daher werden grundlegende Arbeiten der gitterlosen Oberflächenrekonstruktion beschrieben, welche die Basis für die in der letzten Dekade entwickelten Arbeiten darstellen. Dabei werden die beide Hauptrichtungen von gitterlosen Oberflächenrekonstruktion eingeführt, nämlich Oberflächenrepräsentationen basierend auf Projektionsoperatoren und auf impliziten Formulierungen. Die Methoden basierend auf Projektionsoperatoren definieren die approximierte Oberfläche als die Menge statischer Punkte für eine gegebene Abbildung. Die Methoden basierend auf impliziten Formulierungen definieren die approximierte Oberfläche als die Menge von Punkten, die zur Nullmenge der Funktion gehören. Moving-Least-Squares-Oberflächen können sowohl mit Projektionsoperatoren als auch mit impliziten Formulierungen definiert werden, wie später diskutiert werden wird.

Gitterloses Oberflächenrendering wird als nächstes Thema behandelt. Dabei sind Methoden basierend auf Punkten als Renderingprimitive in den letzten Jahren populär geworden. Da sehr viele Arbeiten in diesem Bereich vorgestellt wurden, übersteigt ihre detailierte Behandlung den Umfang dieser Dissertation. Allerdings werden die Hauptideen erwähnt und der Fokus auf Ray-Casting gesetzt, speziell auf die Berechnung des Schnittspunktes zwischen einem Strahl und der gitterlosen Oberfläche. In späteren Kapiteln wird diese Beschreibung im Rahmen der in dieser Arbeit vorgestellten Techniken weiter detailliert.

Nachdem Oberflächenvisualisierung basierend auf der Visualisierungspipeline behandelt wird, wird der gleiche Ansatz für Volumenvisualisierung verwendetet. Die unterschiedlichen Datentypen und Gittertypen, welche von den Visualisierungsmethoden behandelt werden, werden beschrieben. Das breite Spektrum der Gitterypen ist in dieser Arbeit speziell wegen des Bedarfs an unterschiedlichen Rekonstruktionstechniken für unterschiedliche Gittertypen wichtig. Ein Teil der in dieser Dissertation beschriebenen Forschung konzentriert sich auf der Entwicklung einer Technik für die Rekonstruktion von in Gittern mit beliebigen Topologie und Geometrie gespeicherten Volumendaten. Nachdem Rekonstruktionsmethoden beschrieben sind, werden Volumerenderingtechniken dargestellt, speziell direktes Volumenrendering. Dabei wird das Renderingintegral für das Emissions-Absorptions-Modell abgeleitet.

Schließlich endet das Kapitel mit den grundlegenden Konzepten der Grafikhardwareprogrammierung. Die von der meisten Grafikhardware implementierte Renderingpipeline wird beschrieben. Da Grafikhardware in dieser Arbeit oft benutzt wird, um die Berechnungen zu beschleunigen, wird ferner das Konzept von Allzwecksgraphikhardwareprogrammierung eingeführt.

Kapitel 3: Gitterlose Approximationsmethoden

In diesem Kapitel wird ein Überblick über gitterlose Approximationsmethoden gegeben. Die Scattered-Daten-Interpolations- und Approximationsprobleme werden definiert. Dann werden radiale Basisfunktionen beschrieben, wobei das Konzept der radialen Basisfunktionen-Interpolation, sowie die bekanntesten radialen Basis-Funktionen eingeführt werden. Danach wird die polynomiale Moving-Least-Squares-Approximation mit einer einfachen und allgemeinen Definition behandelt.

Zusätzlich zu diesen Grundlagen werden orthogonale Polynome im Rahmen von Moving-Least-Squares beschrieben. Dafür wird das Konzept von Orthogonalität einer Polynombasis für von einer bestimmten Gewichtungsfunktion definierte Skalarprodukte eingeführt. Dann wird eine effiziente Indizierung beschrieben, welche die Anzahl von notwendigen Operationen für die Konstruktion einer orthogonalen Basis mit Gram-Schmidt-Orthogonalisierung reduziert.

Schließlich wird die approximierte Moving-Least-Squares-Approximation als eine effiziente matrizenlose Methode für die Approximation von Scattered-Daten beschrieben. Diese Methode liefert eine Approximation der Lösung eines Moving-Least-Squares-Problems. Die Methode basiert auf spezifischen Generierungsfunktionen, wobei man Lösungen mit verschiedenen Approximationsordnungen erhalten kann. Ferner wurden radiale Basis-Funktionen und approximierte Approximationen mittels eines iterativen Prozesses miteinander verbunden. Dieser Prozess fängt mit einer approximierten Approximation an und konvergiert zu einer radialen Basisfunktionen-Interpolation. Da dieser Prozess in dieser Dissertation verwendet wird, um Oberflächen und Volumendaten zu rekonstruieren, wird ein Überblick über die in dieser Arbeit verwendeten Theorie gegeben.

Kapitel 4: Gitterlose Oberflächen aus Punktmengen

Dieses Kapitel stellt neue Methoden zur Oberflächenrekonstruktion aus Punktmengen vor. Zu Beginn werden bisherige gängige Verfahren erklärt. Projektionsoperatoren werden zuerst behandelt. In diesem Kontext wird ein neuer krümmungsbasierter Projektionsoperator vorgestellt, welcher auf der Berechnung von einem Polynom zweiten Grades basiert. Dieses Polynom wird von den Hauptrichtungen und Krümmungen an einem gegebenen Punkt definiert. Daher wird die robuste Berechnung der Hauptrichtungen und Krümmungen aus Punktmengen gezeigt. Diese Ergebnisse werden dann zusammen mit einer anisotropen Diffusionsgleichung dazu benutzt, um den Projektionsoperator zu definieren. Dieser Operator wird dann in einen Raycaster integriert, um die Oberfläche zu rendern.

Ein weiteres Thema bilden die impliziten Oberflächen. Zwei verschiedene Methoden werden benutzt, um das Problem von Oberflächenrekonstruktion aus Punktmengen zu lösen. Die erste Methode basiert auf Moving-Least-Squares-Oberflächen und behandelt Performanz- und Approximationsprobleme solcher Oberflächen. Approximierte Approximation wird verwendet, um den Rekonstruktionsprozess zu beschleunigen und Approximationen höherer Ordnung zu ermöglichen. Ferner wird der oben genannte iterative Prozess dazu genutzt, interpolierende Oberflächen zu generieren. Die Effizienz dieser Methode wird ausgenutzt, um bilaterale Filterung in den Prozess einzuführen, damit man eine robuste Berechnung von Normalenvektoren erhält. Dies ermöglicht eine bessere Repräsentation von scharfen Kanten, da die Änderungen in dem Normalenfeld wichtiger für die Perzeption von scharfen Kanten als die eigentliche Approximation der Oberfläche sind.

Die zweite in dieser Arbeit vorgeschlagene Methode für die Rekonstruktion von Oberflächen aus Punktmengen basiert auf Partition der Eins. Erstens wird ein Überblick der ursprünglichen Methode gegeben. Diese Methode hat jedoch Probleme mit der Robustheit, was in diesem Kapitel behandelt wird. Ferner ist die hier vorgeschlagene Methode zweifach adaptiv im Sinne, dass der Raum adaptiv geteilt wird und das Grad des Polynomes adaptiv gesetzt wird, um die Approximation den Details der Oberfläche anzupassen. Dies kann mittels orthogonalen Polynomen umgesetzt werden. Allerdings führt dies Probleme der Robustheit ein, welche durch die Entwicklung von sogenannten Domänen-Abdeckungskriteria behandelt werden. Ferner wurde ein interaktives Tool entwickelt, damit der Benutzer die Oberfläche editieren kann, um Fehlern zu korrigieren oder um die Qualität der Approximation zu verbessern. Schließlich wird eine Methode basierend auf

Vertexverschiebungen dazu benutzt, um die Qualität der Dreiecke des resultierenden Gitters zu verbessern.

Im letzten Abschnitt dieses Kapitels wird die Grafikhardware dazu verwendet, um das Rendering von Oberflächen basierend auf Projektionsoperatoren oder impliziten Funktionen zu beschleunigen. Speziell wird ein in der Grafikhardware implementierter Raycaster für gitterlose Oberflächen beschrieben.

Kapitel 5: Gitterlose Oberflächen aus Volumina

Gitterlose Methoden können im Rahmen der Volumensvisualisierung eingesetzt werden. Daher werden in diesem Kapitel zwei verschiedene Visualisierungsprobleme behandelt. Das erste Problem ist die Extraktion von Oberflächen, die eine bestimmte Eigenschaft von einem Skalarfeld repräsentieren. In diesem Fall werden spezifisch Isoflächen und Oberflächen in der Nähe von Bereichen mit hohem Gradientbetrag extrahiert. Dies wird durch die Definition von geeigneten Gewichtungsfunktionen für Moving-Least-Squares-Oberflächen gemacht. Die Domain der Definition wird mittels der Kombination von Moving-Least-Squares und Weighted-Least-Squares erweitert. Dies ermöglicht die Projektion von Punkten, die weit weg von der Oberfläche sind. Der Prozess wird als Grafikhardware-basierter Raycaster implementiert und auf kartesische Gitter angewendet.

Das zweite Problem ist die interaktive Generierung von Strom-Oberflächen und der neuen Pfad-Oberflächen. Das Hauptproblem in der Visualisierung von Vektorfeldern mit Strom-Oberflächen ist der Bedarf an der Triangulierung von der Stromlinien. Dieser Prozess kann langsam sein besonders im Fall von Pfad-Linien. Deshalb werden in dieser Arbeit Strom- und Pfad-Linien mittels einer Grafikhardwareimplementierung generiert. Damit wird die Konstruktion der Strom- und Pfad-Linien schnell genug, um eine interaktive Änderung der Saatpunkte zu ermöglichen. Ferner wird die Dichte der Strom- und Pfad-Linien nach der Auswertung der Integration adaptiv gesetzt, um punktbasierte Renderingmethoden besser zu unterstützen. Dies ermöglicht die Benutzung von Splatting, um die Oberfläche zu rendern und damit wird der Bedarf an Triangulierung vermindert. Line-Integral-Convolution (LIC) wird außerdem verwendet, um die Details der Strömung besser darzustellen.

Kapitel 6: Gitterlose Volumenvisualisierung

Dieses Kapitel führt neue Methoden für die Visualisierung von Gittern mit beliebigen Topologie und Geometrie an. Diese Methoden basieren auf gitterlosen Approximationstechniken, um zugrundeliegende Funktion zu rekonstruieren. Dieses Problem wird mit zwei verschiedenen Methoden behandelt. Die erste Methode basiert auf einer Detail-erhaltenden Approximation der Volumendaten. Dies wird mittels der Minimierung einer spezifischen Funktion gemacht, welche die Details in den Daten erhalten kann. Allerdings ist dieses Problem schlecht konditioniert und da eine iterative Methode aus dieser Formulierung resultiert, wird die Performanz deutlich reduziert. Deswegen werden orthogonale Polynome verwendet, um den Approximationsprozess in jedem Schritt zu beschleunigen, wodurch die Stabilität verbessert wird. Dies ermöglicht die Implementierung der Technik in Grafikhardware, weil kein Gleichungsystem gelöst werden muss.

Eine billigere ungenauere Methode wird auch vorgestellt, basierend auf approximierter Approximation, welche die Berechnungszeit deutlich reduziert. Diese Methode benutzt iterative approximierte Approximation, welche eine radiale Basis-Funktionen-Interpolation approximiert. Außer die Performanz im Vergleich zu Moving-Least-Squares-Approximation und radiale Basis-Funktionen-Interpolation zu verbessern, kann man mit dieser Methode einen Kompromiss zwischen Genauigkeit und Robustheit erreichen. Allerdings muss sich diese Methode an die Anisotropie der Gitter anpassen. Daher werden ellipsenförmige Gewichtungsfunktionen in den Prozess eingeführt und eine neue anisotrope iterative approximierte Approximation definiert. Folglich konvergiert der iterative Prozess zu einer ellipsenförmigen Basisfunktionen-Interpolation. Diese Methode wird auch in Grafikhardware implementiert, um die Funktion im Rahmen eines Raycasters zu rekonstruieren.

Schließlich wird Volumenmanipulation mittels Moving-Least-Squares vorgestellt. Dafür werden Moving-Least-Squares Methoden für die Deformation von Bildern und Oberflächen erweitert, um Volumina zu unterstützen und nicht-lineare polynomiele Deformationen werden eingeführt. Der Hauptpunkt dieser neuen, nicht-linearen Deformationen ist die geschlossene Formulierung, welche einer der Vorteile von Moving-Least-Squares-Deformationen ist. Diese Deformationen wurden in Grafikhardware implementiert, um die Berechnung von Displacement-Maps zu beschleunigen. Daher ist es durch die Definition der Deformationen als Rückwärtsabbildungen möglich, die Displacement-Map zu berechnen, so dass bekannte Grafikhardwarebeschleunigte Volumenrenderingmethoden benutzt werden können. Das Kapitel endet mit einem Vergleich zwischen gitterlosen Deformationen und Grafikhardwarebeschleunigten physikalischen Deformationen für Tetraedernetze. Außer einen Vergleich anbieten zu können, ist die neue Beschreibung der Grafikhardwareimplementierung von impliziten Integrationsmethoden für physikalische Deformationen ein weiterer Beitrag dieser Arbeit.

Kapitel 7: Gitterlose Methoden in der Visualisierung

Das letzte Kapitel dieser Dissertation bietet Richtlinien für die Benutzung der in dieser Arbeit vorgeschlagenen Methoden an. Eine Diskussion über die Vorteile, Probleme und zukünftige Arbeiten kann auch in diesem Kapitel gefunden werden.

CHAPTER 1 INTRODUCTION

Volume visualization has become commonplace in the last years. Different techniques aimed at both improving algorithmic performance and increasing the insight gained from the data have been presented by numerous authors in the visualization community. The range of visualization and manipulation techniques developed to help users gain a better understanding of the volumetric data at hand is very wide. However, independently of the nature of the visualization technique and of the paradigms upon which it is based, a volume model needs to be available.

The volume model is the mathematical abstraction of the acquisition or simulation process that was used to generate the data. However, the method of solution, in most cases, is not attached to the volumetric data and therefore a volume model must be reconstructed from the sampled data, which can be stored at scattered positions or in the elements of a mesh. This model definition is normally based on the sampled data and some kind of neighborhood information, or stencil, that defines which samples have influence upon the reconstructed volumetric data at a given position in the domain. This stencil is usually defined by the k nearest neighbors, in the case of scattered data, and by the neighboring elements, being vertices or cells, in the case of meshes. In most cases this model is chosen without regarding the original method of solution since, as stated before, it is not available. The volume model in interactive visualization methods is usually very simple. For instance, linear interpolation is a popular choice despite the fact that research on reconstruction filters and interpolation/approximation methods that provide a higher-order reconstruction has been reported.

On the other hand, meshless methods in the context of volume data visualization have been restricted to scattered data. However, in the last years, the use of mehless techniques for solving tasks addressed in the past with methods based on combinatorial structures has gained popularity, specially within the surface reconstruction community. Meshless methods provide a number of advantages, such as scalability to a variety of data, independence from explicit connectivity and low storage requirements. Additionally, meshless approximation techniques have proven to be accurate and easy to compute. Theoretical results as well as practical solutions have been reported with success.

Despite the good results obtained with meshless methods for surface approximation, open issues remain to be addressed. Therefore, results from approximation theory are explored in the context of the work reported in this thesis to approach problems presented by meshless surface reconstruction methods. Performance issues due to the computation of local approximations are addressed by means of mathematical results as well as algorithmic solutions implemented on commodity graphics hardware. Furthermore, robustness and numerical issues found in meshless surface approximation methods are addressed and new mechanisms for modeling challenging surface features, such as sharp edges, are provided.

The results obtained in addressing the issues presented by meshless surface reconstruction techniques are then applied to volume data in order to extract surfaces that represent some feature in the volume. Thus, meshless methods for modeling smooth manifolds that approximate iso-surfaces and surfaces located at regions of high gradient magnitude were developed and are presented in this work. Meshless methods are not restricted to modeling, but have been used with increasing popularity in rendering. A large number of works based on local information to render a surface have been proposed. The flexibility of such approach is exploited here to interactively render streamsurfaces and path-surfaces using graphics hardware algorithms.

Following this direction, volume visualization is then addressed using meshless approximation methods. The visualization methods developed are based on a meshless volume model extracted from the data using connectivity information available in the mesh as well as the sampled data stored at the elements of the mesh. Although the visualization methods used as proof of concept in this work are direct volume rendering and iso-surface rendering, the generality of the approximation methods presented allows their use with any visualization technique that needs to reconstruct the underlying function from sampled data. Furthermore, the flexibility of meshless methods allows the use of the proposed techniques with a wide range of meshes. With this, we aim at laying the basis towards defining a general method applicable to a variety of grid types based on meshless techniques that have proven successful in other areas.

1.1 Goals of This Thesis

Since the main application area of meshless methods within computer graphics nowadays is surface reconstruction and rendering from unorganized point clouds, a specific goal of this work is to address performance and approximation issues of known meshless surface approximation techniques. Specifically, to improve the robustness of the surface reconstruction and to accelerate the approximation process without compromising the quality of the reconstruction. This is done to set the basis for the use of surface reconstruction in volume visualization problems. Thus, a further specific goal of this work is to modify the moving least-squares surface approximation method to reconstruct surfaces that represent some feature of interest in the volume, specifically iso-surfaces and surfaces located at regions with high gradient magnitude. Thereby, manifold surfaces representing these features can be obtained. Furthermore, exploiting the advantages of meshless techniques not only for the reconstruction but also for the visualization of surfaces extracted from volume data is sought in this work. This is the goal of the techniques developed to interactively sample and render streamsurfaces and path-surfaces.

As stated before, the main goal of this work is to explore the use of meshless methods for volume visualization. Approximate approximation, orthogonal polynomials and bilateral filtering are used to define meshless methods for reconstructing the underlying function in the data. The specific goal is to obtain an efficient means to reconstruct the function independently of the geometry and topology of the grid. Thus, the focus of the last part of this work is to define a method to compute the function reconstruction required in visualization applications from data stored in meshes of arbitrary type. Details in the data must be preserved and the methods must be easy to understand and to compute. Since the use of these techniques in practical applications is of special importance, the acceleration of the computations by means of hardware implementations of the different techniques proposed is also a specific goal of this work.

1.2 Outline of This Thesis

Chapter 2 provides an introduction to interactive visualization, focused on surface and volume visualization. Popular methods for reconstructing the underlying function of the data are described as well as rendering algorithms for surfaces and volumes. For the latter, we focus on direct volume rendering, specifically for the emission-absorption model. This description is given in the context of the visualization pipeline, which is introduced in the first section. A brief discussion of graphics hardware and the rendering pipeline encloses the chapter.

In Chapter 3, the mathematical background that is the base for the algorithms presented in this work is given. Since the focus of the work is on meshless methods, the chapter is dedicated to offer a general description of moving least-squares, radial basis functions, orthogonal polynomials, and approximate approximation.

The main part of this thesis can be found in Chapters 4 to 6, where meshless methods for modeling and rendering surfaces and volumes are proposed. Chapter 4 is dedicated to meshless methods for surface approximation from point clouds. Issues found in meshless techniques are addressed and contributions to the area in terms of numerical stability, robustness and performance of the methods are presented. The natural extension to this work is the application of meshless surface approximation and rendering methods to surfaces extracted from volumetric data. This is the focus of Chapter 5, where methods to extract smooth two-manifolds from volumetric data and to interactively render streamsurfaces and path-surfaces are proposed.

In Chapter 6, volume visualization based on meshless methods is addressed. Firstly, the problem of devising a volumetric data approximation method, from the visualization point of view, valid for a wide range of meshes and grids, is approached. The use of meshless approximation methods is a clear choice to address this problem since, as opposed to methods based on parameterizations of the position of the evaluation point with respect to the elements of the mesh, they are independent of the mesh connectivity. However, although the approximation method itself is completely meshless, mesh information must be used to influence the approximation obtained so as to include the mesh connectivity information in the computations. The main concern in defining the methods proposed in this chapter is on the accuracy of the approximation, since meshless approximation methods tend to smooth the data; on the performance, since the locality of the computations turns into the need for solving a large number of systems of equations; and on stability. Secondly, interactive meshless volume deformation is addressed by using moving least-squares deformations implemented on the graphics hardware. Affine, similarity, rigid and the novel nonlinear polynomial deformations are addressed.

Finally, Chapter 7 concludes the thesis with an overview of the methods proposed and a discussion of the usefulness of meshless techniques in the visualization. Guidelines on the cases where the techniques presented may be applied are also provided.

1.3 Acknowledgments

I am greatly indebted and most grateful to my advisor, Thomas Ertl, who supported and guided my work, provided a great environment to work in, and was always more than ready to make sure that my stay at the University of Stuttgart, and in Germany in general, be positive and constructive, not only professionally. Many thanks to him. I would like to express my deep gratitude to Daniel Weiskopf and Luis G. Nonato for their valuable and detailed comments and for their help in developing different methods presented in this work.

I thank the German Academic Exchange Service (DAAD) for providing the financial support for carrying out this work. Special thanks to Rosa Nagel and Maria-Luise Nünning for their help with different matters during my stay in Germany.

I am very grateful to João P. Gois, with whom I developed part of the techniques presented in this work; to Antonio Castelo for his useful comments on mathematical issues; to Tobias Schafhitzel, for our work on interactive streamsurfaces and on hardware-assisted rendering of meshless surfaces and volumes, for proofreading Chapter 4, for the long constructive discussions, and for his help on many different matters; to Ralf Botchen, for proofreading Chapter 6, for his valuable help on different matters, and for our still on-going work on higher-order data visualization; to Magnus Strengert and Thomas Klein, for answering an awful amount of technical questions about visualization and graphics hardware, and for proofreading Chapter 2 (Magnus) and Chapter 3 (Thomas); to Tiago Etiene and Valdecir Polizelli-Junior, for their collaboration in developing the curvature-driven projection operator and the adaptive partition of unity implicits; to my advised students, Siegfried Hodri, Clemens Spenrath, Maurice Gilden, and Tjark Bringewat, for the great work; and to João Dihl Comba and Christian Pagot for their hospitality and the pleasure of working with them in Porto Alegre.

Although not included in this thesis, the works on multi-volume fMRI rendering developed with Friedemann Rößler, Markus Knauff and Thomas Fangmeier and on pre-integrated illustrative methods developed with Nikolai Svakhine, David Ebert and Kelly Gaither, were of particular importance to me. To them, and to Martin Kraus for his essential help in making the pre-integrated illustrative techniques work, my deep gratitude. Special thanks to Friedemann for proofreading Chapter 5.

Many thanks to Michel Westenberg, my first office mate, for the many constructive discussions; to Ulrike Ritzmann, for her help with the formalities; and to the persons I had the pleasure to work with at the University of Stuttgart; in particular (in alphabetical order, without the ones mentioned above), Sven Bachthaler, Katrin Bidmon, Rita Borgo, Marianne Castro, Carsten Darchsbacher, Joachim Diepstraten, Mike Eissele, Thomas Engelhardt, Martin Falk, Mark Giereth, Frank Grave, Sebastian Grottel, Rul Gunzenhäuser, Gunter Heidemann, Andreas Hub, Steffen Koch, Sebastian Klenk, Hermann Kreppein, Andreas Langjahr, Dietmar Lippold, Christoph Müller, Thomas Müller, Guido Reina, Matthias Ressel, Dirc Rose, Martin Rotard, Martin Schmid, Waltraud Schweikhardt, Simon Stegmaier, Christiane Taras, Markus Üffinger, Joachim Vollrath, and Manfred Weiler.

The models and datasets used in this work were provided by different persons and organizations. The EtiAnt pointset in Figure 4.1 is courtesy of Tiago Etiene. The Stanford Bunny model in Figures 4.6, 4.10 and 4.18, the Stanford Dragon model in Figures 4.7 and 4.11, the Armadillo Man model in Figure 4.10, and the Lucy model in Figure 4.14 are courtesy of the Stanford Computer Graphics Laboratory. The Skeleton Hand point set in Figure 4.24 is from the Stereolithography Archive at Clemson University. The Neptune model in Figure 4.22, the Fertility and Buste models in Figure 4.26, the Chinese Lion model in Figure 4.21, and the Filigree model in Figure 4.20 are provided by the AIM Shape Repository. The Rocker Arm model in Figure 4.5 is courtesy of Cyberware Inc.

The Knee volume in Figures 4.25 and 6.8 is courtesy of the Department of

Radiology, University of Iowa. The Boston Tee Pot scan in Figures 6.10 and 6.12 is courtesy of Terarecon Inc., MERL and the Brigham and Women's Hospital. The Foot volume in Figure 6.15 is courtesy of Philips Research. The tretrahedral mesh extracted from the Foot dataset is courtesy of Alex Cuadros. The Bucky Ball dataset in Figures 5.1 and 6.4 is courtesy of AVS. The Engine volume in Figures 5.2, 6.2 and 6.7 is courtesy of General Electric. The Cadaver Head volume in Figures 5.2 and 6.7 is courtesy of the North Carolina Memorial Hospital. The Bonsai Tree scans in Figure 6.7 are courtesy of Stefan Röttger.

The Space Shuttle Launch Vehicle dataset in Figures 6.1 and 6.4, the Oxygen Post dataset in Figures 6.5 and 6.6, and the Blunt Fin dataset in Figure 6.6 are provided by the NASA repository. The Combustion Chamber in Figures 6.3 and 6.4 is from the Visualization Toolkit (VTK). The Tornado dataset in Figure 5.3 is courtesy of Roger Crawfis. The Cylinder dataset in Figure 5.6 is courtesy of Octavian Frederich.

It goes without saying that I am most thankful to my family for their constant support. To my wife, Paola Oviedo, many thanks for her love and understanding and for the many useful suggestions to my work.

Eduardo Tejada

Chapter 2 — Interactive Visualization

Visualization is the process performed to provide a graphical depiction of the information contained in a given raw data. The popularization of visualization in the last decades is due, in part, to the increasing complexity of the data available nowadays. Many different data sources exist, which produce usually very complex data of a specific nature. Simulations and data acquisition techniques are able to produce data with different geometry, including point clouds, polygon soups, and polygonal and polyhedral meshes. If the geometric data itself is the input to the visualization process, we deal with computer graphics problems such as surface approximation, mesh healing, and finite-element manipulation. On the other hand, in scientific visualization the geometric data usually has some kind of measured or simulated data attached to the geometric elements. For instance, volumetric data obtained from computer tomography, magnetic resonance imaging, computational fluid dynamics simulations, and sonar equipment, to name a few, are input data with scalar, vectorial or tensorial quantities stored in geometric and topological structures, normally meshes.

In this chapter, an overview of interactive visualization techniques, related to the research reported in this work, is given. Interactive visualization has been the focus of research of a large number of works in the past three decades. Different approaches based on parallel computing, efficient data structures, compression and level-of-detail, signal processing, and hardware-accelerated techniques have been presented. Of particular interest in the context of this thesis are hardware-accelerated techniques, which take advantage of current advances in graphics hardware. Thus, in this chapter, basics concepts on visualization and graphics hardware are given.

2.1 Visualization Pipeline

The *visualization pipeline* describes the stages of the process used to visualize a dataset. Although there are many different versions of the visualization pipeline, they all describe the data flow that transforms the raw data into an image displayed on some device. Figure 2.1 shows the stages of the visualization pipeline, namely, *data acquisition, filtering, mapping* and *rendering*. The intermediate data is also shown.

Data acquisition is comprised by the methods used to produce the *raw data*, such as computer simulations or measurements of natural phenomena. Usually,

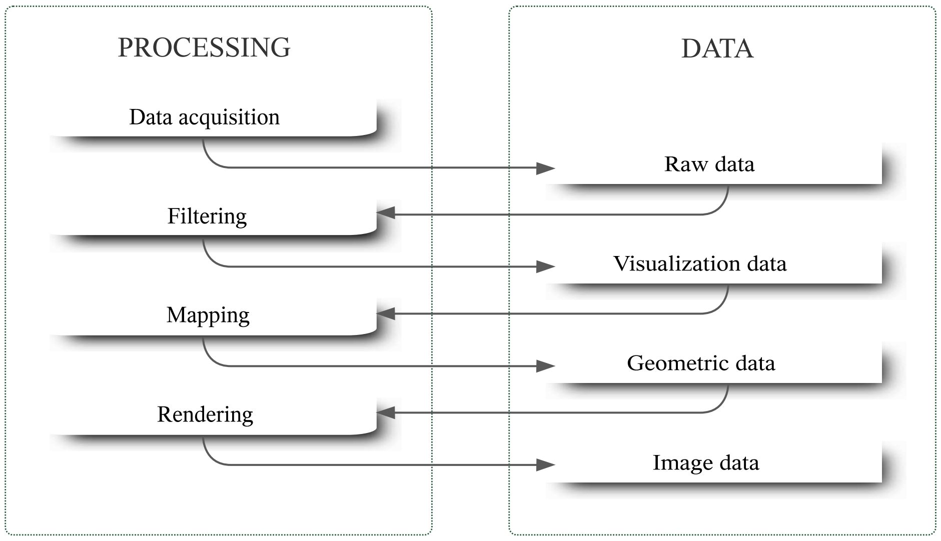


Figure 2.1: Stages and intermediate data of the visualization pipeline.

this raw data is not well-suited for visualization algorithms. The task of processing this data to provide the desired input to the display method is called *filtering*. Filtering includes several operations of different nature, such as interpolation, clipping, and deformation. The data obtained from the filtering, the *visualization data*, is then used to generate a geometric representation during the *mapping* stage. The *geometric data* resulting from the mapping is independent from the method used for displaying it and takes the form of geometric primitives or implicit definitions, as well as properties such as color. The graphical result itself is produced during the *rendering* stage which generates the sought visualization of the data. Although this visualization pipeline arose from the visualization community, it can be directly applied to other areas of computer graphics. This is important for the focus of this thesis, since the flow of the line of research reported here goes from surface modeling and rendering to volume modeling and rendering.

An important filtering task is the reconstruction of the volumetric data at any given point in the domain. As will be seen, interpolation and approximation techniques have been developed to accomplish this task for a wide range of different input data. This filtering partially determines the mapping technique to be used. It is worth mentioning that in the last years, a renewed interest on mapping techniques based on implicit definitions and meshless modeling has arisen, specially in problems related to surface representations. The application of such techniques to volume data has not been fully explored yet. However, their use in this scenario poses new problems for the interactive rendering of the data resulting from mapping techniques based on them. As mentioned before, these are the issues addressed in this thesis.

2.2 Surface Visualization

Although the term surface visualization is often used in scientific visualization applications, some authors use it to refer, in general, to the process described by the visualization pipeline applied to surface data. In this thesis, for sake of consistency, this latter approach is followed to better fit the description of the research reported in Chapters 4 and 5 within the visualization context.

Here, an overview of basic concepts on surface data, surface reconstruction, and surface rendering focused on the approximation of surfaces from sampled data is given. We will see in later chapters that this sampled data can refer both to surface and to volumetric data.

2.2.1 Surface data

In the context of this thesis, input surface data can come from two different sources. The first type of such input data is comprised by the so-called *point clouds* or *unorganized point sets*. A point cloud is a set $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ of points sampled on the surface ∂S of an object $S \subset \mathbb{R}^3$. This sampling is usually performed by means of a three-dimensional scanner (see for instance the Digital Michelangelo Project [100]). Other information, such as the normal vector to the surface at the sample position, radius, and material properties, can be attached to the point data. In this case, each sample is referred to as a *surfel*, short for *surface element*. Currently available technology is able to generate very large point sets with tens and even hundreds of millions of points, usually with problems such as noise, non-uniformity, or regions devoid of data.

Polygon soups are a further type of surface data that has been handled lately with meshless methods. A polygon soup is a set $\mathcal{K}$ of polygons with no inherent structure, *i.e.*, a list of polygons with no connectivity information between them. Sources of polygon soups are often polygonal data sets containing holes, gaps, T-junctions, self intersections, and non-manifold structure. Thus, the term can be used to describe collections of polygons that do not posses guarantees concerning their structure.

Polygonal meshes can also constitute input surface data to meshless methods, for instance, when polygons with bad aspect ratio are found, or up-sampling or down-sampling the surface is the task to perform.

2.2.2 Surface reconstruction

The work on surface reconstruction from sampled surface data is vast and variate. One can broadly divide the approaches found in the literature into meshless approaches and approaches based on combinatorial structures, *i.e.*, mesh-based approaches. In the latter group, approaches using *Delaunay triangulations* [22;

84], *alpha shapes* [19; 43; 15] and *Voronoi diagrams* can be found [8; 9]. These methods usually generate jagged triangle meshes or triangle meshes with poor quality. Therefore, algorithms to smooth the surface and improve the quality of the triangles are often used as post-processing.

On the other hand, meshless approaches do not rely on any combinatorial structure, although the result of the method can be a mesh. Since meshless techniques are the focus of this work, and the mathematical foundation upon which they are based is given in the next chapter, a description of these techniques will be given later throughout the thesis. However, it may be important to mention here two groundbreaking works upon which much current research is based: Hoppe's implicit surface definition [69] and Levin's *moving-least-squares surfaces* [98]. These two works are representatives of the two main approaches to meshless surface reconstruction, namely, the definition of the surface (a) as a level set of an implicit function and (b) as the set of static points for some projection operator. It must be noted, however, that it has been proven that moving-least-squares surfaces can also be stated as an implicit surface [11]. As discussed later, implicit surfaces have a number of advantages, *e.g.*, their suitability for CSG operations or the simplicity of their definition.

Hoppe and collaborators defined the approximate surface as the zero set of a function f_H that approximates the *signed distance* from a point $\mathbf{x}$ to the surface ∂S. A simplicial surface is then constructed by means of a contouring algorithm. The function f_H is defined as

$$f_H(\mathbf{x}) = \langle \mathbf{x} - \mathbf{x}_i, \mathbf{n}_i \rangle,$$

where $\langle , \rangle$ is the scalar product, $\mathbf{x}_i$ is the nearest point to $\mathbf{x}$, and $\mathbf{n}_i$ is the approximation of the normal vector to the surface ∂S at $\mathbf{x}_i$. The vector $\mathbf{n}_i$ is estimated by means of covariance analysis. To that end, for $\mathbf{x}_i$, the covariance matrix

$$C_i = \sum_{\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i)} (\mathbf{x}_j - \mathbf{x}_i) \otimes (\mathbf{x}_j - \mathbf{x}_i)$$

is calculated, where $\mathcal{N}(\mathbf{x}_i)$ is the set of M nearest neighbors to $\mathbf{x}_i$. If λ_k; $k = 1, \cdots, 3$ denote the eigenvalues of C_i, where $\lambda_3 < \lambda_1, \lambda_2$, associated with the eigenvectors $\mathbf{e}_k$; $k = 1, \cdots, 3$, respectively, the vector $\mathbf{n}_i$ is chosen to be either $\mathbf{e}_3$ or $-\mathbf{e}_3$. The actual orientation is computed afterwards by stating the orientation problem as a graph optimization, so as to obtain consistently oriented normal vectors.

On the other hand, Levin defines the *moving least-squares surface* for a point cloud as the set of stationary points of a certain map $f_{MLS} : \mathbb{R}^3 \to \mathbb{R}^3$. Although the *moving least-squares method* is described in the next chapter, the definition of moving least-squares surfaces is given here since the details of moving least-squares are not necessary to understand this surface definition.

Given a point $\mathbf{r} \in \mathbb{R}^3$ near ∂S to be projected, $f_{MLS}(\mathbf{r})$ is defined in two steps as follows. First, a local approximating plane $H = \{\mathbf{x} : \langle \mathbf{a}, \mathbf{x} \rangle - \langle \mathbf{a}, \mathbf{q} \rangle = 0, \mathbf{x} \in \mathbb{R}^3\}$ is computed by finding $\mathbf{q}$ and $\mathbf{a} = \mathbf{a}(\mathbf{q})$; $\|\mathbf{a}\| = 1$, so that $\mathbf{a}$ minimizes

$$e_{MLS}(\mathbf{q}, \mathbf{a}) = \sum_{i=1}^{N} \left(\langle \mathbf{a}, \mathbf{x}_i \rangle - \langle \mathbf{a}, \mathbf{q} \rangle\right)^2 \omega_{MLS}(\mathbf{q}, \mathbf{x}_i) \tag{2.1}$$

where $\mathbf{a}$ is in the direction of the line through $\mathbf{q}$ and $\mathbf{r}$, the directional derivative of $J_{MLS}(\mathbf{q}) = e_{MLS}(\mathbf{q}, \mathbf{a}(\mathbf{q}))$ in the direction of $\mathbf{a}(\mathbf{q})$, evaluated at $\mathbf{q}$ is zero, *i.e.*, $\partial_{\mathbf{a}(\mathbf{q})} J_{MLS}(\mathbf{q}) = 0$, and $\omega_{MLS}(\mathbf{p}, \mathbf{q}) \equiv w(\|\mathbf{p} - \mathbf{q}\|)$, where w is a monotonically decreasing function, typically a Gaussian

$$w(s) = \exp(-\frac{s^2}{h^2}),$$

where h is the fill size. Once H is found, a local polynomial approximation is computed. To that end, let $\{\widetilde{\mathbf{x}}_i : i = 1, \cdots, N\}$ be the orthogonal projections of the points $\{\mathbf{x}_i : i = 1, \cdots, N\}$ onto H, represented in an orthonormal coordinate system on H defined so that $\mathbf{r}$ is projected onto the origin. Also, let $f_i = \langle \mathbf{x}_i, \mathbf{a} \rangle - \langle \mathbf{q}, \mathbf{a} \rangle$; $i = 1, \cdots, N$, be the heights of the points $\{\mathbf{x}_i : i = 1, \cdots, N\}$ over H. Find a polynomial $\widetilde{p} \in \prod_m^2$ as

$$\widetilde{p} = \operatorname*{argmin}_{p \in \Pi_m^2} \sum_{i=1}^{N} \left(p(\widetilde{\mathbf{x}}_i) - f_i\right)^2 \omega_{MLS}(\mathbf{q}, \mathbf{x}_i), \tag{2.2}$$

where Π_m^d is the space of d-variate polynomials of degree m. The projection of $\mathbf{r}$ is defined as

$$f_{MLS}(\mathbf{r}) \equiv \mathbf{q} + \widetilde{p}(\mathbf{0})\mathbf{a}. \tag{2.3}$$

Then,

$$\mathcal{M}\partial S \equiv \{\mathbf{x} \in \mathbb{R}^3 : \mathbf{x} = f_{MLS}(\mathbf{x})\}.$$

Many extensions to these methods have been proposed, among which are those presented in Chapter 4.

2.2.3 Surface rendering

A large number of computer graphics techniques are involved in surface rendering, ranging from visible surface determination to global illumination algorithms. As stated before, the output of the meshless surface approximation method can be used to generate a mesh, in which case traditional rendering methods can be used.

On the other hand, if the meshless surface representation is to be direct input to the rendering method, some considerations must be taken, for instance, for the ray/surface intersection calculation. There is a considerable amount of work on meshless rendering methods and many of them will be described in the following chapters. However, some seminal works on the topic are mentioned here.

Image-order algorithms. Ray-casting (or ray-tracing) meshless surface representations has been approached both for implicit surfaces definitions [2] and for surfaces defined as the set of static points of a projection operator [3]. In the case of implicit surfaces, the surface/ray intersection problem can be regarded as the problem of finding the roots of the implicit function on the domain of the line defined by the ray. Analytical and numerical approaches have been proposed [67]. However, for general implicit functions, the most widely used approach is the bisection method (Figure 2.2), where the ray is sampled with a regular step size starting at a point near ∂S, until a change in the sign of the implicit function is found. Then, the bisection method is applied starting with the two last points (one on each side of the surface). Note that this process is specific to implicit definitions where the inside/outside state is given by the sign of the function.

Computing the intersection of a ray with a surface defined by a projection operator takes a different approach. Starting with a point near ∂S, an iterative process that provides an approximation to the intersection is performed [3]. In each iteration, the projection of the current approximate intersection is computed. If the distance between the point and its intersection (t in Figure 2.2) is less than a predefined threshold, the process ends and the result is the current approximate intersection. Otherwise, a local approximation to the surface, *e.g.*, a polynomial approximation, is computed and its intersection with the ray defines the new approximate intersection. Details on how to compute the local approximation to the surface, on defining its support, and on the data structures used for accelerating the intersection computation are given in Chapters 4 and 5.

Object-order algorithms. A number of techniques are able to generate a dense sampling of points from the original input set of points. Levin's surface definition is an example of this, since the projection operator can be repetitively applied on a dense set of points near ∂S in order to project them onto the approximated surface. When a sufficiently dense sampling is available, *point-based surface rendering* can be applied. The idea behind point-based rendering is to exploit the advantages of points as graphical primitives compared to triangles, namely, the compactness of the representation, ease of manipulation, and flexibility. The first work on point-based rendering was published by Levoy and Whitted [101]. Nowadays, rendering *surfels*, *e.g.*, by means of surface splatting, is the most widely used approach for rendering a dense point set. As stated before, surfels are points which have additional information attached to them, for instance, normal vectors, radii and material properties. This information can be calculated if not available

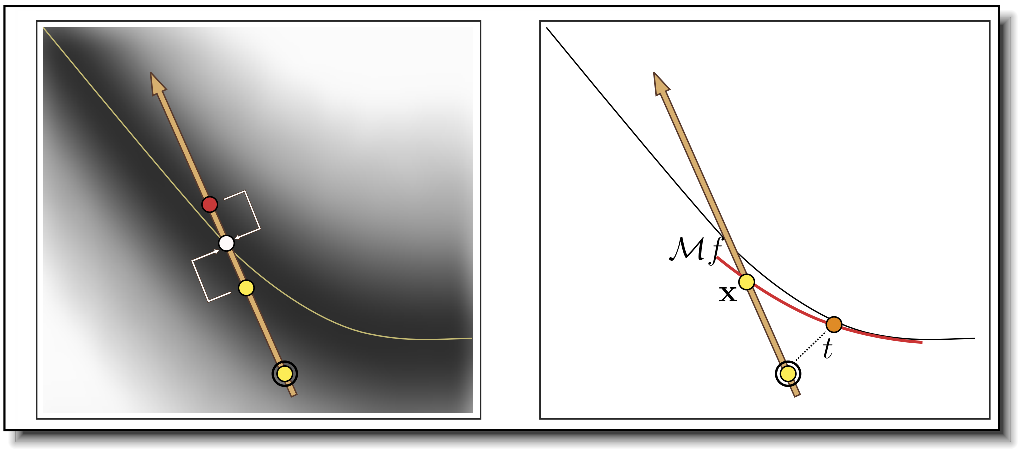


Figure 2.2: Ray-casting implicit surfaces (left) and surfaces defined as the set of static points of some projection operator (right). For implicit definitions, a surface cross is found by sampling the ray at regular intervals. With the two last sampled points, the bisection method is applied to find an approximation to the intersection point. In the case of projection operators, the approximate intersection is projected onto the surface. If the distance t to the projection is less than a threshold, the current approximation is the intersection point. Otherwise a local approximation $\mathcal{M}f$ to the surface is computed and its intersection with the ray defines the new approximate intersection $\mathbf{x}$. In both cases the process starts with a point near the surface ∂S depicted by the circled point (see color plates).

as there is a large amount of work on these topics. The rendering is implemented by projecting the surfels onto the image plane and compositing the contribution of each surfel to the color of the pixels in the projection [126].

2.3 Volume Visualization

The visualization pipeline applied to volumes is known as volume visualization. The goal in volume visualization is to create a graphical representation of the information contained in the volume data to help the user gain insight into it. To that end, many methods that address modeling, filtering and rendering of the volume have been developed. The goal is not only to generate a graphical representation of the data, but to provide the user with a means to better understand it. However, since the literature on volume visualization is very extensive, this section will be focused on basic volume visualization concepts that are used throughout this thesis.

2.3.1 Volume data

Since volume data is acquired by a large number of different means, such as computer simulation and medical imaging, the type, domain and structure of the data,

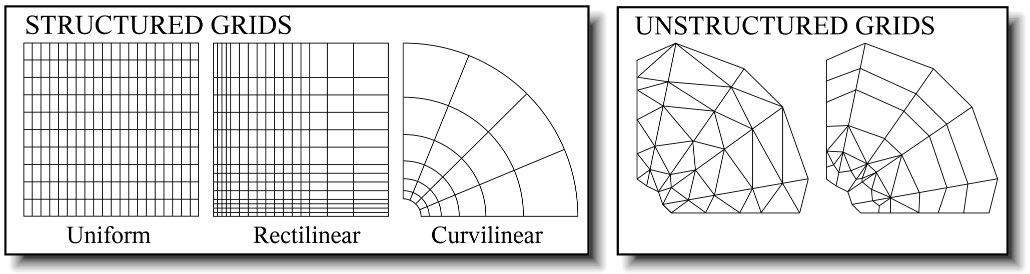


Figure 2.3: Classification of common grids types found in scientific visualization.

in turn, can vary largely. The data type, for instance, ranges from scalar data to higher-order tensor data, and very often it is possible to find volume data with multiple fields of different type.

Concerning the structure, volume data can be stored, in general, as scattered data or at the elements of a grid (Figure 2.3). Grids are usually subdivided in two broad types, namely, *structured* and *unstructured grids*. The sample positions in structured grids can be indexed, for the three-dimensional case, by (i, j, k), where $i \in \mathcal{I}, j \in \mathcal{J}$, and $k \in \mathcal{K}$, and $\mathcal{I}, \mathcal{J}, \mathcal{K}$ denote indexing sets. Among structured grids, *regular grids* possess the most 'regular' geometry and the position of each sample i, j, k, for the three-dimensional case, is implicitly stored and can be reconstructed by

$$\mathbf{v}_{i,j,k} = (i\Delta x, j\Delta y, k\Delta z)\,\mathbf{R} + \mathbf{t},$$

where $\mathbf{R}$ is a rotation matrix, $\mathbf{t}$ is a translation vector, and Δx, Δy and Δz are the grid cell size in each direction. A specific subtype of regular grids are *Cartesian grids,* for which $\Delta x = \Delta y = \Delta z$.

A more general type of structured grids are *rectilinear grids* which possess irregularly spaced vertices in each dimension, so that

$$\mathbf{v}_{i,j,k} = (x(i), y(j), z(k))\,\mathbf{R} + \mathbf{t}.$$

Curvilinear grids have the same connectivity as rectilinear grids however the vertex position cannot be implicitly defined and must be explicitly specified as a position in space

$$\mathbf{v}_{i,j,k} = (x(i,j,k), y(i,j,k), z(i,j,k))\,\mathbf{R} + \mathbf{t}.$$

The advantage of curvilinear grids is that the domain of the simulation can be better represented without having to increase the resolution as it would be the case with rectilinear grids. Also, the implicit nature of the connectivity is maintained. This type of structured grids is largely used in the aircraft and car manufacturing industries.

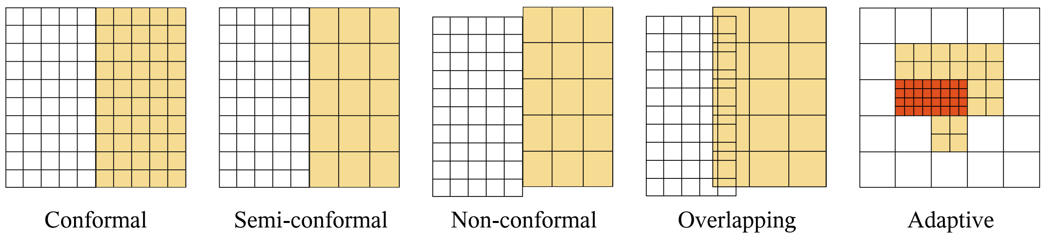


Figure 2.4: Different multiblock grid types.

Unstructured grids are the most general grid type since the connectivity is explicitly stored. These grids are also referred to as polyhedral meshes. A polyhedral mesh is a finite set of polyhedra where the intersection of any two polyhedra is either empty or a face, edge or vertex of each; or for any partition of the set into two subsets, there is always at least one polygon that is a face of a polyhedron from each subset. The term cell and polyhedron will be used synonymously in the following. In practice, the cells in these grids are usually limited to tetrahedra, hexahedra, prisms, and pyramids. As shown in Figure 2.3, unstructured grids can have cells of different type.

Multiblock grids (Figure 2.4) are also often found as result of computer simulations. Multiblock grids are formed by two or more grids of any of the previously mentioned grid types. Multiblock grids can be classified as *conformal*, *semi-conformal*, *non-conformal* and *overlapping grids*. Conformal grids are the easiest to handle, since a natural continuity in the connectivity is present. This is also the case for semi-conformal grids, where the main issue that must be addressed is the different 'resolutions' of the grids. Non-conformal grids, on the other hand, can be arbitrarily placed with the only constraint that the intersection of two grids is non-void and the meshes do not overlap as is the case with overlapping grids. A very important grid type that can be included among the multiblock grids is comprised by the *adaptive mesh refinement* grids. These grids are formed by a set of grids, where the grids of greater resolution are within the coarser grids and the boundaries of the former are identical to the boundaries of the cell in the latter that contains it. This type of grid adapts the resolution of each block to the accuracy requirement of each part of the domain. Interpolating in multiblock grids is a challenging task, which is addressed in Chapter 6.

2.3.2 Volume data reconstruction

In general, visualization algorithms require as input a reconstruction of the volume data from the samples on the entire domain. This data reconstruction is usually associated with interpolation methods. However, approximation methods can also be useful, for instance, when noisy data is to be handled. Volume data reconstruction is also referred to as *filtering*, specially in reconstruction methods

for Cartesian grids. Here, an overview of the most common approaches for volume data reconstruction is given. In the next chapter, meshless approximation methods that are directly related to the research reported in this thesis will be described in detail. As will be seen in Chapter 6, these methods can be used for volume data reconstruction in arbitrary grids, while providing a means to define a unified approach that is able to generate both interpolations and approximations of the volume data. That is, a method that is able to effectively deal with data with and without noise stored in arbitrary meshes will be presented. In the following, let $\mathcal{X}$ be the set of sample points $\mathbf{x}_i \in \mathbb{R}^3$; $i = 1, \ldots, N$, in a three-dimensional domain. The sample points can be the vertices of a grid, the cell centers of a grid or scattered samples. The sampled data at sample point $\mathbf{x}_i$ is referred to by f_i.
Scattered data. In general, there are two basic approaches to scattered data reconstruction. The first is based only on the scattered positions while the second makes use of some sort of spatial decomposition to aid the interpolation process. In the latter approach, once the spatial decomposition is computed, the values are reconstructed using reconstruction methods for gridded data, such as the described below. On the other hand, meshless scattered data reconstruction methods, as mentioned above, are applied directly on the scattered samples. Since meshless data interpolation and approximation are directly related to the research reported here, a detailed description that includes methods based on partition of unity, least-squares, and radial basis functions will be given in the next chapter as part of the mathematical foundations. However, here the two most widely known reconstruction methods for meshless scattered data reconstruction are briefly described, namely, Shepard's method [143], also known as *inverse distance weighing,* and Sibson's interpolation [7]. Shepard's interpolation can be written as

$$f_s(\mathbf{x}) = \frac{\sum_{i=1}^{N} f_i \omega_s(\mathbf{x}, \mathbf{x}_i)}{\sum_{i=1}^{N} \omega_s(\mathbf{x}, \mathbf{x}_i)},$$

where

$$\omega_s(\mathbf{x}, \mathbf{y}) = \frac{1}{\|\mathbf{x} - \mathbf{y}\|^2}.$$

Shepard's method produces C^0-continuous interpolations and cusps, corners and flat spots can be obtained. Modifications to this method that address this issue have been proposed, which fall within the category of methods based on partition of unity. Shepard's method can be regarded as the simplest case of the moving least-squares method.

On the other hand, data interpolation using Sibson's parameterization produces C^1-continuous interpolations. Sibson's method takes into account only the *natural neighbors* of the evaluation point $\mathbf{x}$ to calculate the interpolated value. The *Voronoi diagram* F of the set $\mathcal{X}$ is a domain partitioning into regions $V(\mathbf{x}_i)$, such that any point in $V(\mathbf{x}_i)$ is closer to *site* $\mathbf{x}_i$ than any other site $\mathbf{x}_j$. The regions $V(\mathbf{x}_i)$ are called *Voronoi cells*. Given $\mathcal{F}$ and $\widehat{\mathcal{F}}$, where $\widehat{\mathcal{F}}$ is the Voronoi diagram of the set $\mathcal{X} \cup \{\mathbf{x}\}$, with Voronoi cells $\widehat{V}(\mathbf{p}); \mathbf{p} \in \mathcal{X} \cup \{\mathbf{x}\}$, the set $\mathcal{N}_{\widehat{\mathcal{F}}}(\mathbf{x})$ of natural neighbors of $\mathbf{x}$ is comprised by the sites of the neighboring Voronoi cells to the cell of the site $\mathbf{x}$. The Sibson's interpolant is then calculated as

$$f_c(\mathbf{x}) = \frac{\sum_{\mathbf{x}_i \in \mathcal{N}_{\widehat{\mathcal{F}}}(\mathbf{x})} f_i \omega_c(\mathbf{x}, \mathbf{x}_i)}{\sum_{\mathbf{x}_i \in \mathcal{N}_{\widehat{\mathcal{F}}}(\mathbf{x})} \omega_c(\mathbf{x}, \mathbf{x}_i)},$$

where $\omega_c(\mathbf{x}, \mathbf{y}) = v(\widehat{V}(\mathbf{x}) \cap V(\mathbf{y}))$ and v is a function that returns the volume of a region.

Rectilinear grids. The simplest reconstruction method used in rectilinear grids is the *nearest-neighbor interpolation*, where the reconstructed value $f_n(\mathbf{x})$ at the evaluation point $\mathbf{x}$ is given by f_i, where $\|\mathbf{x}_i - \mathbf{x}\| < \|\mathbf{x}_j - \mathbf{x}\|; \forall j \neq i$. Since this interpolation is discontinuous, unpleasant abrupt changes in the visual representation (rendering) are obtained. On the other hand, piecewise tri-linear interpolation generates C^0-continuous reconstructions and due to its simplicity, ease of coding, and low computational cost, is widely used in visualization methods. Given the vertices $\mathbf{x}_{c_k}; k = 1, \ldots, 8$, of the cell C, such that $\mathbf{x}$ is in the interior of C, the reconstructed value $f_t(\mathbf{x})$ is in this case obtained as

$$\begin{aligned} f_t(\mathbf{x}) \quad = \quad & (1-\alpha)\,(1-\beta)\,(1-\gamma)\,f_{c_1} + \alpha\,(1-\beta)\,(1-\gamma)\,f_{c_2} \\ & + \alpha\,\beta\,(1-\gamma)\,f_{c_3} + (1-\alpha)\,\beta\,(1-\gamma)\,f_{c_4} \\ & + (1-\alpha)\,(1-\beta)\,\gamma\,f_{c_5} + \alpha\,(1-\beta)\,\gamma\,f_{c_6} \\ & + \alpha\,\beta\,\gamma\,f_{c_7} + (1-\alpha)\,\beta\,\gamma\,f_{c_8} \end{aligned}$$

with

$$\alpha = \frac{\|\widehat{\mathbf{x}}_{c_1c_2} - \mathbf{x}_{c_1}\|}{\|\mathbf{x}_{c_2} - \mathbf{x}_{c_1}\|}, \quad \beta = \frac{\|\widehat{\mathbf{x}}_{c_1c_4} - \mathbf{x}_{c_1}\|}{\|\mathbf{x}_{c_4} - \mathbf{x}_{c_1}\|}, \quad \text{and} \quad \gamma = \frac{\|\widehat{\mathbf{x}}_{c_1c_5} - \mathbf{x}_{c_1}\|}{\|\mathbf{x}_{c_5} - \mathbf{x}_{c_1}\|},$$

where $\widehat{\mathbf{x}}_{c_1c_2}$, $\widehat{\mathbf{x}}_{c_1c_4}$, and $\widehat{\mathbf{x}}_{c_1c_5}$ are the projections of $\mathbf{x}$ on the lines defined by the pairs of vertices $(\mathbf{x}_{c_1}, \mathbf{x}_{c_2})$, $(\mathbf{x}_{c_1}, \mathbf{x}_{c_4})$, and $(\mathbf{x}_{c_1}, \mathbf{x}_{c_5})$ respectively. If the data is cell centered, the dual mesh is used.

Higher-order reconstruction schemes such as *B-splines*, *Catmull-Rom splines* and, *windowed* sinc *filters* are also used in visualization applications [109]. These schemes are *separable filters*, which can be written as $h(\vartheta, \varrho, \varsigma) = h_s(\vartheta) h_s(\varrho) h_s(\varsigma)$. The reconstructed value is obtained as the sum over the sampling points,

$$f_h(\mathbf{x}) = \sum_{i=1}^{N} f_i h(\vartheta_i, \varrho_i, \varsigma_i),$$

where $(\vartheta_i, \varrho_i, \varsigma_i) = \mathbf{x}_i - \mathbf{x}$. Catmull-Rom splines and cubic B-splines belong to the family of cubic splines

$$h_s(\varsigma) = \frac{1}{6}\begin{cases} (6-2b) - (18-12b-6c)|\varsigma|^2 + (12-9b-6c)|\varsigma|^3 & |\varsigma| < 1 \\ 8(b+3c) - 12(b+4c)|\varsigma| + 6(b+5c)|\varsigma|^2 - (b+6c)|\varsigma|^3 & 1 \le |\varsigma| < 2 \\ 0 & \text{otherwise,} \end{cases}$$

where different b and c generate different cubic splines. Note that this assumes that the distances between two neighboring samples in any dimension is one. If this is not the case, a normalization in each dimension is performed. B-splines are obtained by setting $b = 1$ and $c = 0$, while Catmull-Rom splines are obtained with $b = 0$ and $c = 0.5$. Windowed sinc filters, on the other hand, approximate the ideal sinc filter with a filter of finite support. Since simple truncation of the sinc filter causes ringing artifacts, it is multiplied with functions that drop smoothly at the boundaries of the support. The defining equation of a windowed sinc, considering one window, is

$$h_s(\varsigma_m)(\varsigma) = \begin{cases} (1 + \cos(\pi\varsigma/\varsigma_m))\,\mathrm{sinc}(4\varsigma/\varsigma_m) & |\varsigma| < \varsigma_m \\ 0 & \text{otherwise,} \end{cases}$$

where ς_m is the radius of the support. Note that this filter needs to be normalized to ensure the unity of its integral on the domain. As a side note, tri-linear interpolation is also a separable filter.

Lagrange interpolation can also be found to a lesser extent in visualization methods. In the univariate case, the approach fits a polynomial P of degree $d-1$ to d points $(\mathbf{x}_j, f_j); j : 1, \ldots, d$, as

$$P(\mathbf{x}) = \sum_{j=1}^{d} P_j(\mathbf{x}),$$

where

$$P_j(\mathbf{x}) = f_j \prod_{\substack{k=1 \\ k \neq j}}^{d} \frac{\mathbf{x} - \mathbf{x}_k}{\mathbf{x}_j - \mathbf{x}_k}.$$

The method can be extended to the three-variate case for rectilinear grids to obtain a reconstruction $f_l(\mathbf{x})$ of the value at $\mathbf{x}$ by successively applying the interpolation to higher dimensions.

Curvilinear and unstructured grids. Linear interpolation in unstructured and curvilinear grids is also widely used in visualization applications. The most common approach is to partition each polyhedral cell into tetrahedra and compute the linear interpolation on the tetrahedral partition. Note, however, that this interpolation depends on the partition used. Given a point $\mathbf{x}$ in the interior of a tetrahedron $T = [\mathbf{x}_{v_1}, \mathbf{x}_{v_2}, \mathbf{x}_{v_3}, \mathbf{x}_{v_4}]$ in the tetrahedral mesh, the barycentric coordinates $\lambda_k; k = 1, \ldots, 4$, of $\mathbf{x}$ with respect to T are obtained by solving

$$\sum_{k=1}^{4} \lambda_k \mathbf{x}_{v_k} = \mathbf{x},$$

$$\sum_{k=1}^{4} \lambda_k = 1.$$

Then, the volumetric data can be linearly interpolated within T as

$$f_t(\mathbf{x}) = \sum_{k=1}^{4} \lambda_k f_{v_k}.$$

Barycentric coordinates are a parameterization of the position of a point with respect to the tetrahedra. Similarly, *mean value coordinates* where recently proposed by Floater *et al.* [52] to parameterize any star-shaped polyhedra. Let $\Omega \subset \mathbb{R}^3$ be the domain defined by the set of cells of a polyhedral mesh. Furthermore, let $C \subset \Omega$ be a polyhedron in the mesh, with triangular facets and vertices $\mathbf{x}_{v_k}$; $k = 1, \ldots, m$. The kernel K of C is the open set consisting of all points $\mathbf{x}$ in the interior of C with the property that for all $k = 1, \ldots, m$, the only intersection between the line segment $[\mathbf{x}, \mathbf{x}_{v_k}]$ and the boundary ∂C is $\mathbf{x}_{v_k}$. C is *star-shaped*, if K is non-empty. The boundary ∂C of C is a mesh of triangles. For any $\mathbf{x}$ in K, each oriented triangle $H = [\mathbf{x}_{v_j}, \mathbf{x}_{v_k}, \mathbf{x}_{v_m}]$ defines a tetrahedron $T = [\mathbf{x}, \mathbf{x}_{v_j}, \mathbf{x}_{v_k}, \mathbf{x}_{v_m}]$. Given the angle β_{rs}; $r, s = 1, \ldots, 3$, between two line segments $[\mathbf{x}, \mathbf{x}_{v_r}]$ and $[\mathbf{x}, \mathbf{x}_{v_s}]$, and the unit normal $\mathbf{n}_{rs}$ to the face $[\mathbf{x}, \mathbf{x}_{v_r}, \mathbf{x}_{v_s}]$, pointing into the tetrahedron T, the barycentric coordinates of $\mathbf{x}$ with respect to C are defined by

$$\lambda_j = \frac{w_j}{\sum_{k=1}^{m} w_k},$$

where

$$w_j = \frac{1}{r_j} \sum_{H \ni \mathbf{x}_{v_j}} \upsilon_{j,H},$$

$r_j = \|\mathbf{x} - \mathbf{x}_j\|$ and

$$\upsilon_{j,H} = \frac{\beta_{km} + \beta_{jk} \langle \mathbf{n}_{km}, \mathbf{n}_{km} \rangle + \beta_{mj} \langle \mathbf{n}_{mj}, \mathbf{n}_{km} \rangle}{2 \langle \mathbf{e}_j, \mathbf{n}_{km} \rangle},$$

where

$$\mathbf{e}_j = \frac{\mathbf{x}_{v_j} - \mathbf{x}}{\|\mathbf{x}_{v_j} - \mathbf{x}\|}.$$

Higher-order approximation in tetrahedral meshes is usually addressed with the *Bernstein-Bézier form* of trivariate splines [39]. Therefore, a tetrahedral partition must be constructed. Let Δ be a tetrahedral partition of Ω in $\mathbb{R}^3$. Then for any integers $0 \le r \le d$, the associated space of polynomial splines of degree d and smoothness r is defined by

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

where $\mathcal{P}_d$ is the space of trivariate polynomials of degree d. Although there is no general theory, there are a few C^1 trivariate spline spaces which have been shown to be useful in applications, such as the classical finite-element spaces with $d = 9$ on general tetrahedral partitions and finite-element spaces with $d = 5$, $d = 3$, and $d = 2$ on subpartitions of Δ where every tetrahedron is split into four, twelve, and twenty four tetrahedra respectively.

Given a tetrahedron $T = [\mathbf{x}_{v_1}, \mathbf{x}_{v_2}, \mathbf{x}_{v_3}, \mathbf{x}_{v_4}]$, any cubic spline s on Δ, can be written in its piecewise Berstein-Bézier form

$$s|_T = \sum_{i+j+k+l=d} c_{ijkl} B_{ijkl}^d, \; c_{ijkl} \in \mathbb{R},$$

where c_{ijkl} are called the *Bernstein-Bézier coefficients* of the polynomial piece $s|_T$ associated with the *Bézier points*

$$\left\{ \xi_{ijkl}^T = \frac{i\mathbf{x}_{v_1} + j\mathbf{x}_{v_2} + k\mathbf{x}_{v_3} + l\mathbf{x}_{v_4}}{d} \right\}_{i+j+k+l=d}.$$

Here, the *Bernstein basis polynomials* of degree d with respect to T are

$$B_{ijkl}^d(\mathbf{x}) = \frac{d!}{i!j!k!l!} \lambda_1^i \lambda_2^j \lambda_3^k \lambda_4^l, \; i + j + k + l = d.$$

If $s \in C^r(\Omega)$ with $r \ge 1$, then the coefficients c_{ijkl} must satisfy certain smoothness conditions. Suppose that two tetrahedra, $T = [\mathbf{x}_{v_1}, \mathbf{x}_{v_2}, \mathbf{x}_{v_3}, \mathbf{x}_{v_4}]$ and $\widetilde{T} = [\mathbf{x}_{v_5}, \mathbf{x}_{v_2}, \mathbf{x}_{v_3}, \mathbf{x}_{v_4}]$, share the face $F = [\mathbf{x}_{v_2}, \mathbf{x}_{v_3}, \mathbf{x}_{v_4}]$. Suppose also that

$$s|_T = \sum_{i+j+k+l=d} c_{ijkl} B_{ijkl}^d,$$

$$s|_{\widetilde{T}} = \sum_{i+j+k+l=d} \widetilde{c}_{ijkl} \widetilde{B}_{ijkl}^d,$$

where $\{\widetilde{B}^d_{ijkl}\}_{i+j+k+l=d}$ are the Bernstein polynomials of degree d associated with $\widetilde{T}$. Given $1 \leq i \leq d$, let

$$\tau^i_{jkl} = c_{ijkl} - \sum_{\nu+\mu+\kappa+\iota=i} \widetilde{c}_{\nu,j+\mu,k+\kappa,l+\iota}\widetilde{B}^i_{\nu\mu\kappa\iota}(v_1).$$

for all $j + k + l = d - i$. Note that for a given pair of adjoining tetrahedra, τ^i_{jkl} is uniquely associated with the domain point ξ^T_{ijkl}. The spline s is C^r continuous across the face F if and only if

$$\tau^i_{jkl}s = 0, \text{ for all } j + k + l = d - i \text{ and } i = 0, ..., r.$$

Thus, the reconstructed value at $\mathbf{x} \in T$ is $f_b(\mathbf{x}) = s|_T(\mathbf{x})$.
Multiblock grids. Reconstructing the volume data stored in multiblock grids is a challenging task depending on the grid type. Conformal multiblock grids can be addressed with the approaches described above. Semi-conformal, non-conformal, and adaptive-mesh-refinement grids can be addressed by reconstructing the value inside each cell independently if linear interpolation suffices. However, discontinuities are often introduced. Overlapping multiblock grids pose an even more difficult problem, since the domains of two or more grids overlap. As mentioned before, this problem is addressed in Chapter 6.

2.3.3 Volume rendering

Volume rendering is the process in the visualization pipeline that generates the graphical representation of a volume data set. Volume rendering methods can be broadly divided into *direct volume rendering* methods and *indirect volume rendering* methods. The former regard the volume data as a non-opaque cloud of particles, and visualize it directly by modeling certain physical effects, while the latter visualize some derived model, *e.g.*, an *isosurface mesh*. Since indirect volume rendering in the context of this thesis, as will be seen in the subsequent chapters, is in the form of surface visualization, the above presented description of surface rendering suffices. Therefore, direct volume rendering is the focus of the following discussion.

Direct volume rendering aims at visualizing a volume by modeling three physical effects affecting the appearance of non-opaque materials: *emission*, *absorption* and *scattering*. Emission refers to the light coming from chemical reactions or excitation on an atomic level. In a scattering process a photon interacts with a scattering center and emerges from the event moving in a different direction in general with a different frequency (*inelastic scattering*). If the frequency does not change, one speaks of *elastic scattering*. Absorption refers to the attenuation of light between a particle and a light source. In the following, the focus of the discussion is on the so-called *emission-absorption model*, which will be used

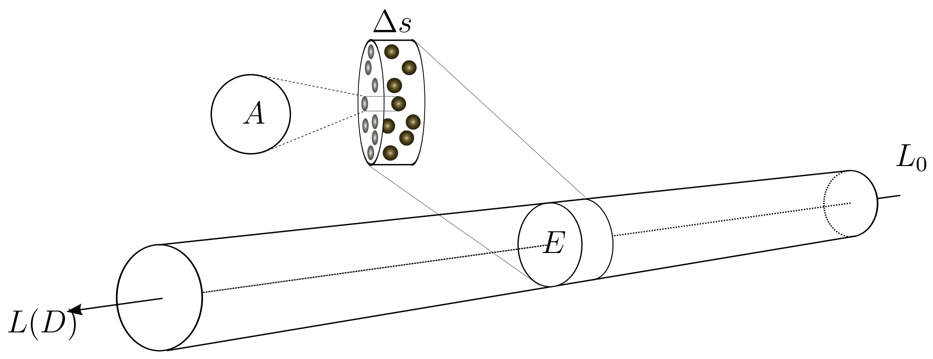


Figure 2.5: Ray through the volume.

throughout the rest of the thesis. A detailed description of different models including scattering, shadowing and multiple scattering can be found in the work by Max [111].

Volume rendering integral. Consider a cylinder with length D centered around a viewing ray that passes through the volume, whose radius is small enough so that volume properties change only along its length (Figure 2.5). Also, consider a light source positioned at the end of the cylinder, *i.e.* the extreme opposite to the view point, with radiance L_0 (per wavelength) in the direction of the ray. The color of the pixel corresponding to the ray is determined by the radiance $L(D)$ coming from the front of the cylinder. Consider furthermore a thin slab of this cylinder with base area E and length Δs. Finally, consider a participating medium with particle density ρ. In the emission-absorption model, as the light ray flows along the direction Δs the particles absorb the light that they intercept and emit new light. For simplicity, assume that all particles are identical spheres with radius r. The area of the projection of each particle on the base of the slab is $A = \pi r^2$. The volume of the slab is $E\Delta s$ and therefore contains $N = \rho E \Delta s$ particles. If Δs is small enough, the particle projections on the base of the slab have low probability of overlap. Thus, the area of the base of the slab occluded by the particles is approximated by $NA = \rho A E \Delta s$ and the fraction of light occluded when flowing through the slab is

$$A\rho E\Delta s / E = \rho A \Delta s. \tag{2.4}$$

As stated above, in addition to absorbing light, the particles emit light with intensity C per unit projected area. Thus, the radiance of the light emitted by the N particles in the slab is $CAN = C\rho A E \Delta s$ which gives an added flux per unit area equal to

$$C\rho A \Delta s. \tag{2.5}$$

Thus, considering the light occluded (Equation 2.4) and the light emitted (Equation 2.5) by the particles in the slab per unit area, since as Δs approaches zero, the probability of overlap also approaches zero, the change in the radiance of a ray of light through the volume can be defined as

$$\begin{aligned}\frac{dL}{ds} &= C(s)\rho(s)A - \rho(s)AL(s) \\ &= C(s)\tau(s) - \tau(s)L(s) \\ &= L_e(s) - \tau(s)L(s). \end{aligned} \tag{2.6}$$

The quantity $\tau(s)$ is called the *extinction coefficient* and $L_e(s)$ is called the *source term*.

To solve this differential equation, the term $\tau(s)L(s)$ is brought to the left hand side and both sides are multiplied by the integrating factor $\exp\left(\int_0^s \tau(t)dt\right)$ giving

$$\left(\frac{dL}{ds} + L(s)\tau(s)\right)\exp\left(\int_0^s \tau(t)dt\right) = L_e(s)\exp\left(\int_0^s \tau(t)dt\right)$$

or

$$\frac{d}{ds}\left(L(s)\exp\left(\int_0^s \tau(t)dt\right)\right) = L_e(s)\exp\left(\int_0^s \tau(t)dt\right).$$

Integrating from $s = 0$, at the back end of the volume, to $s = D$, at the eye, we obtain

$$L(D)\exp\left(\int_0^D \tau(t)dt\right) - L_0 = \int_0^D L_e(s)\exp\left(\int_0^s \tau(t)dt\right)ds,$$

which can be rewritten as

$$L(D) = L_0\exp\left(-\int_0^D \tau(t)dt\right) + \int_0^D L_e(s)\exp\left(-\int_0^s \tau(t)dt\right)ds. \tag{2.7}$$

An analytic solution of this integral is, in general, not feasible. Thus, numerical integration is needed. The most common numerical approximation of the volume rendering integral is done by means of Riemann sums as follows. The interval from 0 to D is divided into n equal segments of length $\Delta x = D/n$ and a sample x_i at each segment is choses to be $x_i = i\Delta x$. Then

$$\begin{aligned}\exp\left(-\int_0^D \tau(x)dx\right) &\approx \exp\left(-\sum_{i=1}^{n}\tau(i\Delta x)\Delta x\right) \\ &= \prod_{i=1}^{n}\exp\left(-\tau(i\Delta x)\Delta x\right) = \prod_{i=1}^{n} t^{(i)}.\end{aligned}$$

Let $L_e^{(i)} = L_e(i\Delta t)$ and define

$$\exp\left(-\int_{i\Delta x}^{D} \tau(x)dx\right) \approx \prod_{j=i+1}^{n} t^{(j)}.$$

The Riemann sum for

$$\int_0^D L_e(s)\exp\left(-\int_0^s \tau(t)dt\right)ds$$

becomes

$$\sum_{i=1}^{n} L_e^{(i)} \prod_{j=i+1}^{n} t^{(j)},$$

and the final estimate of Equation 2.7 is

$$\begin{aligned} L(D) &\approx L_0 \prod_{i=1}^{n} t^{(i)} + \sum_{i=1}^{n} L_e^{(i)} \prod_{j=i+1}^{n} t^{(j)} \\ &= L_e^{(n)} + t^{(n)}\left(L_e^{(n-1)} + t^{(n-1)}\left(L_e^{(n-2)} + \cdots \left(L_e^{(1)} + t^{(1)}L_0\right)\cdots\right)\right), \end{aligned}$$

which gives us the back-to-front compositing formulation

$$L^{(i)} = L_e^{(i)} + (1-\alpha^{(i)})L^{(i-1)},$$

where $L^{(i)}$ is the accumulated color for the i first ray segments, $\alpha^{(i)} = 1 - t^{(i)}$ can be thought of as the opacity of the i-th segment along the ray, and $L^{(0)} = L_0$. $L_e^{(i)}$ is know as the *pre-multiplied color* or *associated color* in volume rendering algorithms.

A better approximation to the volume rendering integral can be obtained by using *pre-integrated classification* [85], where a continuous, piecewise linear scalar function is reconstructed along the viewing ray and the volume rendering integral between each pair of successive samples of the scalar field is evaluated by table look-ups.

Image-order algorithms. The evaluation of the volume rendering integral is common to all direct volume rendering algorithms. As in the case of surface rendering, these algorithms can be subdivided into object-order and image-order algorithms. Ray-casting techniques for volume rendering [99; 54] are image-order algorithms, where usually at least one ray is traced for each pixel in the image from the view point to the volume. The volume is traversed along each ray and the final color is composited using the color and opacities reconstructed at each sample point on the ray. In Figure 2.6 the ray-casting process is depicted. The

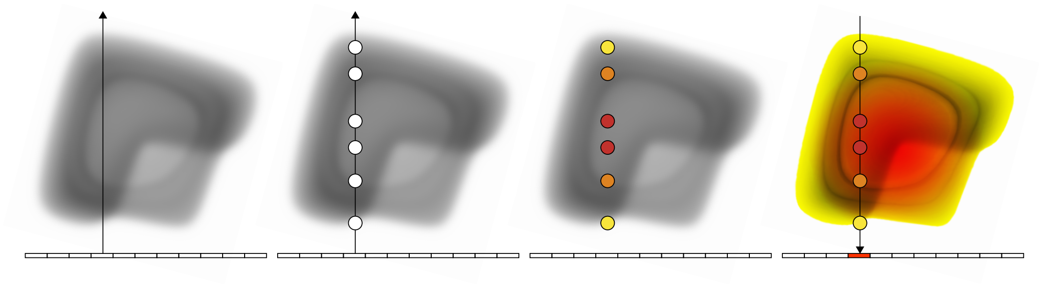


Figure 2.6: Volume ray-casting (see color plates). From left to right: at least one ray is traced for each pixel in the image (ray casting), on each ray the volume is sampled a number of times (sampling), the contribution of each segment is computed (shading), and the contributions of all ray segments are composited to determine to final color of the pixel (composition).

four main steps are shown, namely, *ray casting*, *sampling*, *shading*, and *compositing*. During ray casting, the rays for each pixel are traced from the view point to the volume. Each ray is then sampled at specific locations (sampling). The sampling locations are not necessarily evenly spaced and depend largely on the grid type and on the volume ray-casting techniques implemented, *e.g.*, acceleration techniques and feature enhancement methods. The way the contribution of each ray segment is computed can also vary due to the specific shading technique used and to the numerical approximation of the volume integral for the individual segment. The contributions of all ray segments are then blended together during the *compositing* as explained above. In the last years, one of the main problems approached by researches has been the acceleration of the rendering process by means of graphics hardware [45].

Object-order algorithms. In contrast to image-order algorithms, object-order volume rendering algorithms compute the contribution of individual parts of the volume to the rendering integral. Usually, each part contributes to the integral along many rays. One of the most widely used image-order algorithms is *cell projection* where cells in the grid are projected onto the image plane in visibility order (see Figure 2.7). The contributions of the cells to the integral along a ray are blended using compositing as described above. Another very well known object-order algorithm is *splatting* [171], which generates the image by computing for each *voxel* (a cell in a Cartesian grid together with its volumetric data) its contribution to the result in all pixels which overlap with its *footprint* in image space. The computation is performed by slicing the volume and project each slice onto the image plane. The footprint of each voxel is actually the reconstruction kernel centered around it (see the section on volume data reconstruction above).

The support for texture mapping by graphics hardware made possible the de-

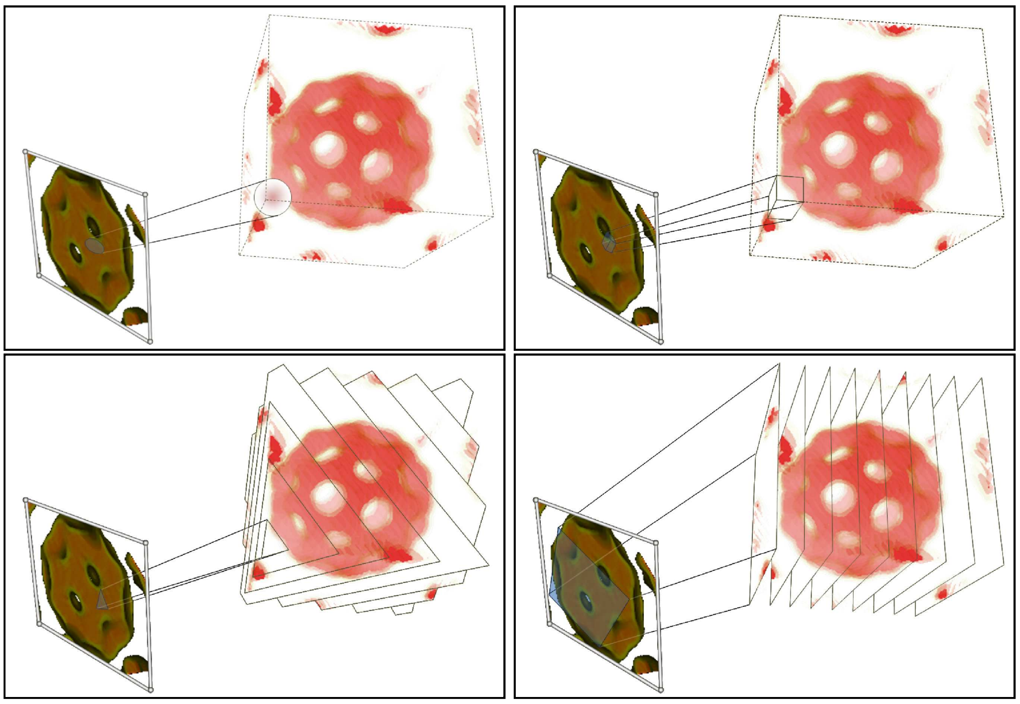


Figure 2.7: Object-order volume rendering algorithms. Clockwise (from top left): splatting, cell projection, object-aligned, and view-aligned texture-based volume rendering. Note that splatting and cell projection are not restricted to regular grids (see color plates).

velopment of *texture-based volume rendering* methods for Cartesian grids [25] (Figure 2.7). Where two-dimensional texturing is available, volume rendering can be accomplished by projecting object-aligned textured slices onto the image plane. The contribution of the slices to the final image are composited together by means of the blending operations supported by the graphics hardware. In this case, three stacks of slices aligned along the axis of the object coordinate system are used. During rendering, the stack corresponding to the axis closest to the viewing direction is used. This requirement of maintaining three stacks of textures is alleviated with the support for three-dimensional textures, in which case the volume is sliced orthogonally to the viewing direction. The resulting slices are textured directly by accessing the three-dimensional texture holding the volume. An approach related to texture-based volume rendering is the *shear-warp algorithm* [87] where the viewing transformation is factorized such that the nearest facet of the volume becomes axis aligned with an off-screen image buffer with a fixed scale of voxels to pixels. The volume is then renderered into this buffer using this more favorable memory alignment. Once all slices of the volume have been rendered, by rasterizing them in software, the buffer is then warped into the desired orientation and scale in the displayed image.

2.4 Visualization and Graphics Processing Units

In the last years, the rapid technological advances in graphics hardware has been exploited in different areas of computer graphics to accelerate processes that accomplish tasks not necessarily restricted to those for which the graphics processing units (GPU) are designed. This includes the implementation on graphics hardware of many visualization algorithms for modeling and rendering volume data sets, which has provided interactive means for the real-time exploration and manipulation of simulated and measured data.

The capabilities of commodity graphics hardware have been exploited for accelerating the techniques presented in this thesis. Therefore, in this section, basic concepts of commodity graphics hardware and their use in general programming tasks will be introduced.

2.4.1 The rendering pipeline

The process performed by the graphics hardware to display a list of primitives is known as the *rendering pipeline* (see Figure 2.8). This list of primitives, generated by the application, is input to the rendering pipeline in the form of vertices. These vertices are then transformed, first to *world coordinates*, then to *eye coordinates*, *clip coordinates*, *normalized device coordinates*, and finally to *windows coordinates*, by the vertex processor (which in Figure 2.8 includes the primitive assembly stage). The spatial relationships among the local coordinates systems of the objects are defined in world coordinates. The per-vertex lighting computations and the specification of texture coordinates are also performed in world coordinates. Eye coordinates result from positioning a virtual camera at an arbitrary location in world coordinates. The transformation to eye space is determined by the position, the viewing direction and the up vector of the camera. Culling is performed in eye coordinates since the visibility of the polygons is determined by the line-of-sight and the normal vector and center of each polygon. The vertices in eye coordinates are then projected onto the viewing plane to obtain a two-dimensional representation of the scene in clip coordinates. In this space, clipping and removal of hidden surfaces are performed. Perspective division is then performed to obtain normalized device coordinates. The normalized device coordinates are then transformed to window coordinates by means of a viewport transformation. Finally, in the last stage of the rendering pipeline, the projected polygons are rasterized. Rasterization is the process by which the representation of the polygons in window coordinates is converted into raster format, vertex attributes are interpolated and per-*fragment* shading (including texturing) and hidden surface removal is performed. A *fragment* can be regarded as a pixel with additional information such as depth and texture coordinates.

Since the introduction of the *programmable pipeline*, the operations performed

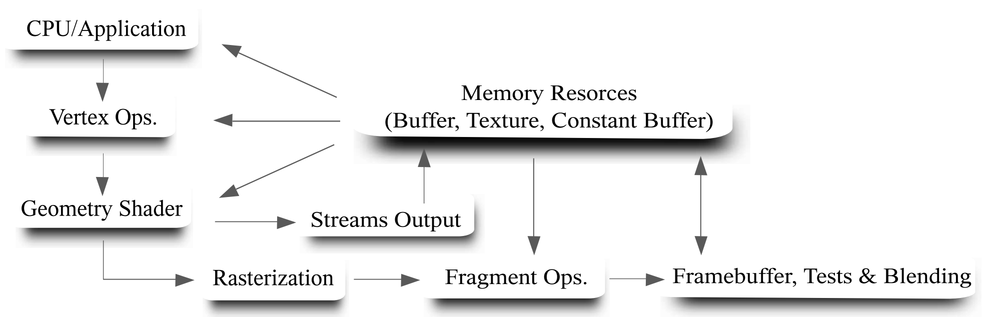


Figure 2.8: The programmable rendering pipeline.

by the vertex and fragment processors can be defined by the programmer by writing GPU programs. Additionally, the geometry shader stage has been introduced into the pipeline recently. A *vertex program* is a graphics processing function that performs mathematical operations on the vertex data, *e.g.*, geometrical transformations, per-vertex lighting and texture coordinates computations. It is important to note that the operations are performed on the vertex data input to the rendering pipeline by the application or on data previously written by the *stream output stage*, later described. A *geometry shader*, on the other hand, can generate new primitives from existing primitives. The geometry shader is executed after the vertex program shader and its input is the primitive with adjacency information. During the stream output stage, the vertex data generated by the geometry shader is streamed out to buffers in graphics memory, always as complete primitives but without the adjacency information. As hinted before, this streamed data can be read back into the pipeline, which would be processed in a subsequent rendering pass. Similarly, the operations performed by the fragment processor, can be programed by means of *fragment programs*. These operations are performed on a per-fragment basis.

One important aspect of the rendering pipeline is that the GPU can only process independent vertices and fragments. However, it can process many of them in parallel. That is, the GPU is in some sense a stream processor and the programs (kernels) perform the same operations on all elements (vertices, primitives, fragments) in parallel. This fact has been exploited by many authors to perform tasks for which the GPU was not originally designed. This is done by mapping the programming problem at hand into a stream processing problem in order to exploit the high parallelism and computational power of the GPU, as will be seen in the following.

2.4.2 General-purpose GPU programming

The use of the GPU for solving general computing problems is known as *general-purpose computations on the GPU* (GPGPU). Since GPUs are designed to perform graphics tasks, their programing is very restrictive compared to the CPU. However, problems that can be solved using stream processing can be effectively tackled using the GPU. In this context, the programmable processors of the GPU are seen as resources that perform the operations defined by the kernels. The capabilities of the rasterizer for creating fragments and interpolating per-vertex constants are also often exploited in GPGPU applications and the recent introduction of the geometry shader has given programmers a new tool with new capabilities for solving problems. The texture unit and the framebuffer can be seen as read-only and write-only memory interface respectively. A write-only texture can be attached to the framebuffer in order to store results in it. Another possibility is given by the recently introduced stream-out stage. Since the introduction of *Shader Model 3*, branching and loops are supported by the processors. However, such flow control structures have a significant performance penalty. Currently available hardware supports single floating point precision in all its stages, as well as in the texture unit. This has boosted the used of GPUs in non-graphical applications.

Recently, in order to overcome the limitations of the use of graphics hardware in stream processing, NVIDIA® and ATI® developed the Compute Unified Device Architecture (CUDA$^{\text{TM}}$) and Close To Metal$^{\text{TM}}$(CTM) technologies respectively. CUDA allows the use of the computing features of the latest NVIDIA GPUs through the standard C programming language. That is, the shader kernels are replaced by kernels written in C. The advantage of using CUDA on NVIDIA GPUs is the possibility of processing thousands of threads simultaneously in comparison with multi-core CPUs that can execute only a few threads at the same time. Additionally, the threads on NVIDIA GPUs can communicate among themselves.

CTM, on the other hand, is a hardware interface that gives developers access to the native instruction set and memory of the AMD Stream Processors and its Radeon GPUs. The use of CTM opens up the architecture of the stream processors and offers developers the low-level, deterministic, and repeatable access to hardware that is essential to develop compilers, debuggers, mathematical libraries, and applications platforms.

CHAPTER

3 MESHLESS APPROXIMATION METHODS

Meshless approximation methods were originally used to solve geoscience problems. Eventually, applications in other areas such as PDEs, artificial intelligence, signal processing, sampling theory, and optimization started making use of these tools. This popularization was due to the need of standard methods, such as splines, for an underlying mesh to define the basis functions. The most widely known multivariate meshless method, often referred to as *inverse distance weighting*, was introduced by Shepard [143]. Hardy [64] proposed the *multiquadric* and *inverse multiquadric* methods, while Duchon [42] developed *thin plate splines*. Lancaster and Šalkauskas [89; 88] generalized the idea of Shepard's functions to define the *moving least-squares* method. On the other hand, the amount of research on *radial basis functions* is vast. However, the work by Wendland [168] is of special significance, since Wendland presented for the first time a class of compactly supported radial basis functions. This made possible the use of computationally efficient meshless radial basis function methods.

In this chapter, a brief overview of radial basis functions and the moving least-squares method is given. Also, the use of *orthogonal polynomials* and *approximate approximation* within the context of moving least-squares is described. The definitions presented here are used throughout this thesis to propose meshless modeling and visualization techniques.

3.1 Radial Basis Functions

The general approximation problem can be defined as follows. Given a set of points $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\} \subsetneq \mathbb{R}^s$, and a function $f : \mathbb{R}^s \to \mathbb{R}$ evaluated on $\mathcal{X}$, generating the set $\mathcal{F} = \{f_1, \ldots, f_N\} \subsetneq \mathbb{R}$, find a function $\mathcal{M}f$ such that $\mathcal{M}f(\mathbf{x}_j) \approx f_j$; $j = 1, \cdots, N$, that is, a function $\mathcal{M}f$ that approximates the data $(\mathbf{x}_j, f_j)$; $j : 1, \ldots, N$.

On the other hand, in the case of interpolation, the function $\mathcal{M}f$ must hold $\mathcal{M}f(\mathbf{x}_j) = f_j$. An usual approach to solve both the approximation and the interpolation problems is to regard the function $\mathcal{M}f$ as a linear combination of certain *basis functions* β_k:

$$\mathcal{M}f(\mathbf{x}) = \sum_{k=1}^{N} c_k \beta_k(\mathbf{x}), \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^s$. Solving the interpolation problem using this approach leads to a system of linear equations of the form $\mathbf{Ac} = \mathbf{f}$, where $\mathbf{A} = \{\beta_k(\mathbf{x}_j)\}_{j,k=1}^N$, $\mathbf{c} = [c_1, \cdots, c_N]^t$ and $\mathbf{f} = [f_1, \cdots, f_N]^t$. Thus, a unique solution to the problem exists if and only if $\mathbf{A}$ is non-singular. It is known that a *positive definite* matrix is non-singular. Thus, since a *strictly positive definite function* is a function f such that for any real numbers $x_1, \cdots, x_N$, the $N \times N$ matrix $\mathbf{M} = \{f(x_i - x_j)\}_{i,j=1}^N$ is a positive definite matrix, to ensure that $\mathbf{A}$ is non-singular, one can choose a set of strictly positive definite functions $\beta_k(\mathbf{x}) = \psi(\mathbf{x} - \mathbf{x}_k)$ as basis functions. Thus, the function $\mathcal{M}f$ is a *translation invariant* interpolant.

A function $\psi : \mathbb{R}^s \to \mathbb{R}$ is called *radial* if there exists a univariate function $\sigma : [0, \infty) \to \mathbb{R}$ such that

$$\psi(\mathbf{x} - \mathbf{x}_k) = \sigma(r),$$

where $r = \|\mathbf{x} - \mathbf{x}_k\|$. Thus, radial functions are not only invariant under translation but also under rotation and reflection. More importantly, the approximation problem becomes insensitive to the dimension s.

There is a variety of basis functions that have been studied and applied in computer graphics, among which are *thin plate splines*, *multiquadrics* and *Gaussian* functions. Thin plate splines were presented by Duchon [42], who defined the radial function as

$$\sigma(r) = \begin{cases} r^2 \log(r) & r \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The function σ is the fundamental solution of the biharmonic equation $\Delta^2 \widetilde{\sigma}(\mathbf{x}) = 0$ where Δ is the *Laplace operator*.

Multiquadrics and inverse multiquadrics where introduced by Hardy. The multiquadric and inverse multiquadric basis functions are defined as

$$\sigma(r) = \sqrt{a^2 + r^2}$$

and

$$\sigma(r) = \frac{1}{\sqrt{a^2 + r^2}},$$

respectively, for some constant a.

Gaussian functions of the type

$$\sigma(r) = \exp(-\epsilon r^2),$$

where ϵ is a constant, are the most widely used basis functions in computer graphics applications, perhaps due to their smoothness and rapid decay despite serious drawbacks such as their sensitiveness to the choice of the parameter ϵ.

3.2 Moving Least-Squares

For a set of basis functions (polynomial functions throughout this thesis) $\Psi = \{\psi_1, \ldots, \psi_M\}$, the moving least-squares method [88] aims at defining a linear combination of Ψ that approximates f. Let us define the vectors $\Gamma = [f_1, \cdots, f_N]$ and $\Psi_j = [\psi_j(\mathbf{x}_1), \ldots, \psi_j(\mathbf{x}_N)]$. Let us also define the inner product $\langle , \rangle_\omega : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}_+$ as a weighted sum:

$$\langle \xi, \eta \rangle_\omega(\mathbf{x}) = \sum_{i=1}^{N} \xi_i \eta_i \omega(\mathbf{x}, \mathbf{x}_i), \tag{3.2}$$

where $\omega(\mathbf{p}, \mathbf{q}) \equiv w(\|\mathbf{p} - \mathbf{q}\|)$, $w : \mathbb{R} \to \mathbb{R}_+$ being a monotonically decreasing function. Note that ω depends on the evaluation point $\mathbf{x}$ and $\mathcal{X}$. A function $\mathcal{M}f$ that minimizes

$$\min \mathbf{E}(\mathbf{x}) = \sum_{i=1}^{N} \left(f_i - \mathcal{M}f(\mathbf{x}_i)\right)^2 \omega(\mathbf{x}, \mathbf{x}_i), \tag{3.3}$$

where

$$\mathcal{M}f(\mathbf{x}) = \sum_{j=1}^{M} c_j(\mathbf{x}) \psi_j(\mathbf{x})$$

is known as *moving least-squares* (MLS) polynomial approximation because as $\mathbf{x}$ changes, the minimization changes, *i.e.*, the solution depends on the evaluation point.

It is known [38] that $\mathcal{M}f$ can be found by solving the system:

$$\begin{bmatrix} \langle \Psi_1, \Psi_1 \rangle_\omega & \ldots & \langle \Psi_1, \Psi_M \rangle_\omega \\ \vdots & \ddots & \vdots \\ \langle \Psi_M, \Psi_1 \rangle_\omega & \ldots & \langle \Psi_M, \Psi_M \rangle_\omega \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_M \end{bmatrix} = \begin{bmatrix} \langle \Gamma, \Psi_1 \rangle_\omega \\ \vdots \\ \langle \Gamma, \Psi_M \rangle_\omega \end{bmatrix} \tag{3.4}$$

or compactly written

$$\left\{ \sum_{j=1}^{M} \langle \Psi_i, \Psi_j \rangle_\omega c_j = \langle \Gamma, \Psi_i \rangle_\omega ;\ i = 1, \ldots, M \right. .$$

3.3 Orthogonal Polynomials in Moving Least-Squares

If a set Ψ is defined such that the inner product satisfies $\langle \Psi_i, \Psi_j \rangle_\omega = \kappa_{ij} \delta_{ij}$, where δ_{ij} is the Kronecker delta, System 3.4 becomes a linear system where the coefficient matrix is diagonal. This means that Ψ is a set of orthogonal polynomials with respect to the inner product. Thus, the moving least-squares approximation is given by the sum

$$\mathcal{M}f(\mathbf{x}) = \sum_{i=1}^{M} \psi_i(\mathbf{x}) \frac{\langle \Gamma, \Psi_i \rangle_\omega}{\langle \Psi_i, \Psi_i \rangle_\omega}. \tag{3.5}$$

A set Ψ with such property can be obtained by making use of the multivariate Gram-Schmidt orthogonalization process. However, this process is computationally unattractive. On the other hand, the revised Gram-Schmidt process for several variables by Weisfeld [164] provides a generalization of the recurrence of three terms [24] for polynomials of several variables, making it a more attractive method in terms of computational performance. Bartels and Jezioranski [18] improved the results by Weisfeld and presented an even more efficient method. The authors argue that the revised Gram-Schmidt process is more efficient than calculating and solving System 3.4. Although the method by Bartels and Jezioranski is used in this thesis, there are different ways to construct orthogonal polynomials in several variables, since the construction is not ensured to be unique. Methods to construct the set Ψ can be found in the works by Stokman *et al.* [147] and by Philips [127]. Here, the method by Bartels and Jezioranski is used since it always produces a matrix-free (or system-free) moving least-squares approximation, *i.e.* no systems of equations must be solved, and is easy to understand and to implement for discrete domains without losing efficiency. Therefore, this method is described below for $s = 3$.

3.3.1 Indexing orthogonal polynomials

In the following bold formatting (*e.g.*, $\mathbf{x}$) will be used for points in $\mathbb{R}^3$ and indexed normal formatting (*e.g.* x_i) for the components of the point, *i.e.*, $\mathbf{x} = (x_1, x_2, x_3)$. The approach by Bartels and Jezioranski to construct a set of orthogonal polynomials Ψ is based on a special ordering of a set $\{x_1^{s_1} x_2^{s_2} x_3^{s_3} : s_i \in \mathcal{I} \subsetneq \mathbb{N}, i = 1, 2, 3\}$ of multinomials and a mapping of such ordered multinomials to integer numbers so as to reduce the number of operations performed. This method is described here for the three-variate case for sake of clarity.

Let us arrange the multinomials in a recursive triangular pattern where the r-th row contains all multinomials of $(r-1)$-th power and each row, with the exception of the first, is organized into 3 groups (ranges): row (1) contains the multinomial 1; row (2) contains 3 ranges with multinomials x_1, x_2, x_3; and row (r) has as their 1st, 2nd and 3rd ranges the multinomials found by multiplying x_1, x_2 and x_3 by each member, in order, of ranges $1, 2, 3$; $2, 3$ and 3 in row $r - 1$ respectively. For instance the second, third and fourth rows of the triangular array are (recall that the first row contains only the monomial 1):

Row 2. $\underbracket{x_1}\ \underbracket{x_2}\ \underbracket{x_3}$
Row 3. $\underbracket{x_1^2\ x_1x_2\ x_1x_3}\ \underbracket{x_2^2\ x_2x_3}\ \underbracket{x_3^2}$
Row 4. $\underbracket{x_1^3\ x_1^2x_2\ x_1^2x_3\ x_1x_2^2\ x_1x_2x_3\ x_1x_3^2}\ \underbracket{x_2^3\ x_2^2x_3\ x_2x_3^2}\ \underbracket{x_3^3}$

where the symbol $\underbracket{\quad}$ indicates a range. The set of positions in this table is $\{i\} = \{1, 2, 3, 4, 5, 6, \dots\}$, that is, *e.g.*, the multinomial in position 6 is x_1x_2. The set of multinomials is $\{\beta(i)\} = \{1, x_1, x_2, x_3, x_1^2, x_1x_2, x_1x_3, x_2^2, x_2x_3, \dots\}$ and the set of vectors of exponents associated with $\{\beta(i)\}$ is given by $\{\gamma(i)\} = \{(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (2, 0, 0), (1, 1, 0), (1, 0, 1), \dots\}$.

During the construction of the orthogonal polynomials, it is important to track polynomials previously constructed. Therefore, the predecessor j' of position j in the triangular array is defined as follows: for the first range, we have $\gamma(j) = (j_1, j_2, j_3)$ which is in the row $j_1 + j_2 + j_3 + 1$. The position of the predecessor j' of j in the array can be found by considering $\gamma(j') = (j_1 - 1, j_2, j_3)$. Thus by looking for $\gamma(j')$ in the set $\{\gamma(i)\}$ it is possible to identify the position j' (which is in row $(j_1 - 1) + j_2 + j_3 + 1$). For instance, let us consider $j = 6$, which is in the first range. Thus, $\beta(6) = x_1x_2$ whose $\gamma(6) = (1, 1, 0)$ and $\gamma(j') = (0, 1, 0)$. Therefore, $j' = 3$. For the second range, similar arguments can be followed: $\gamma(j) = (0, j_2, j_3)$ which is in the row $j_2 + j_3 + 1$. The position of j' can be found by considering $\gamma(j') = (0, j_2 - 1, j_3)$. The predecessor j' is in this case in row $(j_2 - 1) + j_3 + 1$. For the third range, $\gamma(j) = (0, 0, j_3)$ which is in the row $j_3 + 1$. Thus, j' is found by considering $\gamma(j') = (0, 0, j_3 - 1)$, which is in row $(j_3 - 1) + 1$.

Based on this ordering, the construction of the set of orthogonal polynomials is performed following the same triangular pattern so as to let ψ_i correspond to position i in the array.

3.3.2 Constructing orthogonal polynomials

The revised Gram-Schmidt process is given by the following recurrence relation

$$\psi_1 = 1 \text{ and } \psi_j = x_{k_j}\psi_{j'} - \sum_{l=1}^{j-1} \alpha_{j,l}\psi_l; j = 2, 3, \dots,$$

where $x_{k_j} = x_1$ when the predecessor j' is in the first range, $x_{k_j} = x_2$, when j' is the second range and $x_{k_j} = x_3$ when j' is in the third range,

$$\alpha_{j,l} = \frac{\langle x_{k_j}\Psi_{j'}, \Psi_l\rangle_\omega}{\langle \Psi_l, \Psi_l\rangle_\omega},$$

and $\langle x_{k_j}\Psi_{j'}, \Psi_l\rangle_\omega = \sum_{i=1}^{N} x_{k_j}{}^{[i]}\psi_{j'}{}^{[i]}\psi_l^{[i]}\omega^{[i]}$, where the upper index $[i]$ means the function evaluated at point $\mathbf{x}_i$ and $\psi_{j'}$ is the j'-th orthogonal polynomial.

The traversal of the table of multinomials is performed in the following way. As j runs through one range k_j $(1, 2, \text{ or}, 3)$, j' runs from range k_j to 3 in row $r-1$ (note that in both cases there are the same number of multinomials). When j jumps from range k_j to k_j+1, j' is set back to the beginning of range k_j+1 in row $r-1$. A similar process is also considered for the predecessor j'' of j'.

The first six orthogonal polynomials calculated with this process have the following structure: $\psi_1(\mathbf{x}) = 1$, $\psi_2(\mathbf{x}) = x_1 + a$, $\psi_3(\mathbf{x}) = x_2 + b\ x_1 + c$, $\psi_4(\mathbf{x}) = x_3 + d\ x_2 + e\ x_1 + f$, $\psi_5(\mathbf{x}) = x_1\psi_2(\mathbf{x}) + g\ x_3 + h\ x_2 + i\ x_1 + j$, and $\psi_6(\mathbf{x}) = x_1\psi_3(\mathbf{x}) + k\ x_1^2 + l\ x_3 + m\ x_2 + n\ x_1 + o$, where a through o are constants depending on $\mathbf{x}$ and $\mathcal{X}$.

Here, the three-variate polynomials ψ_m, for $m = 2, \cdots, 10$, obtained with this method are shown (recall that $\psi_1(\mathbf{x}) = 1$.)

$$
\begin{aligned}
\psi_2(\mathbf{x}) &= x_1 - \frac{\sum_{i=1}^n x_1^{[i]}\omega^{[i]}}{\sum_{i=1}^n \omega^{[i]}} \\
\psi_3(\mathbf{x}) &= x_2 - \sum_{l=1}^{2} \frac{\sum_{i=1}^n x_2^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x}) \\
\psi_4(\mathbf{x}) &= x_3 - \sum_{l=1}^{3} \frac{\sum_{i=1}^n x_3^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x}) \\
\psi_5(\mathbf{x}) &= x_1\psi_2(\mathbf{x}) - \sum_{l=1}^{4} \frac{\sum_{i=1}^n x_1^{[i]}\psi_2^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x}) \\
\psi_6(\mathbf{x}) &= x_1\psi_3(\mathbf{x}) - \sum_{l=1}^{5} \frac{\sum_{i=1}^n x_1^{[i]}\psi_3^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x}) \\
\psi_7(\mathbf{x}) &= x_1\psi_4(\mathbf{x}) - \sum_{l=1}^{6} \frac{\sum_{i=1}^n x_1^{[i]}\psi_4^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x}) \\
\psi_8(\mathbf{x}) &= x_2\psi_3(\mathbf{x}) - \sum_{l=1}^{7} \frac{\sum_{i=1}^n x_2^{[i]}\psi_3^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x}) \\
\psi_9(\mathbf{x}) &= x_2\psi_4(\mathbf{x}) - \sum_{l=1}^{8} \frac{\sum_{i=1}^n x_2^{[i]}\psi_4^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x}) \\
\psi_{10}(\mathbf{x}) &= x_3\psi_4(\mathbf{x}) - \sum_{l=1}^{9} \frac{\sum_{i=1}^n x_3^{[i]}\psi_4^{[i]}\psi_l^{[i]}\omega^{[i]}}{\sum_{i=1}^n \psi_l^{[i]}\psi_l^{[i]}\omega^{[i]}}\psi_l(\mathbf{x})
\end{aligned}
$$

3.3.3 Avoiding repetitive computations

To reduce the computational cost, two important results by Bartels and Jezioranski are used : (1) $\forall i < j'' \Rightarrow \alpha_{j,i} = 0$ where j'' is the predecessor of j', and (2) if

j, l, p and m are such that $\langle x_{k_j}\psi_{j'}, \psi_l\rangle_\omega = \langle x_{k_p}\psi_{p'}, \psi_m\rangle_\omega$, then

$$\alpha_{j,l} = \alpha_{p,m}\frac{\langle\Psi_m, \Psi_m\rangle_\omega}{\langle\Psi_l, \Psi_l\rangle_\omega}.$$

The former helps to identify the α's with value equal to 0 and the latter makes it possible to reuse inner products previously computed.

3.4 Approximate Approximation

Approximate moving least-squares (AMLS) [46] is a computationally efficient approach free of systems of equations that is able to achieve higher-order approximations. However, this method provides good approximations only for regularly spaced points. On the other hand, radial basis functions interpolation methods are known to produce good results in many applications. However, numerical instabilities arise during the computation of radial basis functions interpolations due to the nature of the system of equations that must be solved. This system is in general large and ill-conditioned.

Recently, the *iterated approximate moving least-squares* method [48] was developed to overcome the issues mentioned above of both approximate moving least-squares and radial basis functions. This method generates a sequence of approximated solutions that converges to a radial basis functions interpolation. This turns the method suitable for irregularly spaced sample points. Since it is based on approximate approximations, no systems of equations must be solved and thus numerical instabilities are better avoided.

3.4.1 Approximate moving least-squares approximation

Maz'ya [112] proposed higher-order approximation methods free of systems of equations, which achieve approximations up to a certain saturation error which can be negligible due to computer precision [46]. Maz'ya developed such approximate approximations for the numerical solution of differential operators, such as multidimensional integro-differential operators [137]. Fasshauer observed that the approximate approximation by Maz'ya can be regarded as a constrained moving least-squares [46]. In addition, he presents methods to efficiently define such an approximation, therefore named approximate moving least-squares approximation. Although it is a promising theory which can be useful in several applications due to its computational simplicity, to our knowledge, only Fasshauer [47; 46] makes use of the theory in practical problems, specifically data compression.

Consider, as before, a set of points $\mathcal{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^s$ and the set $\mathcal{F} = \{f(\mathbf{x}_0), \ldots, f(\mathbf{x}_N)\}$ of values of some function $f \in C^d$ evaluated on $\mathcal{X}$. Fasshauer [47; 46] defines the approximate moving least-squares approximation, for a regularly spaced set of points $\mathcal{X}$, as the function $\mathcal{M}f$ that approximates f as

$$f(\mathbf{x}) \approx \mathcal{M}f(\mathbf{x}) = \sum_{i=1}^{N} f_i \varphi_i(\mathbf{x}), \tag{3.6}$$

for suitable pre-defined *generating functions* φ_i. For instance, Fasshauer proposes:

$$\varphi_i(\mathbf{x}) = \frac{\epsilon^s}{\prod^{s/2}} L_d^{s/2}\left(r_i(\mathbf{x})\right) \exp\left(-r_i(\mathbf{x})\right), \tag{3.7}$$

where $r_i(\mathbf{x}) = \epsilon^s \|\mathbf{x} - \mathbf{x}_i\|^2 / h^2$, $L_d^{s/2}$ are the generalized Laguerre polynomials of degree d [167], h is the fill distance of the data set, s is the dimension, and ϵ is the shape parameter that controls the saturation error by scaling the basic weight function. Using such generating functions, also known as Laguerre-Gaussian functions, it is possible to ensure an approximation order of $O(h^{2d+2})$.

Although there are important results extending this method to irregularly spaced points, they are difficult to implement [46]. However, Fasshauer and Zhang [48] proved recently an interesting result that helps overcome such deficiency. Specifically, the authors proved a connection between approximate moving least-squares and radial basis functions interpolation. It was shown that it is possible to obtain a radial basis functions interpolation using an iterative process based on approximate moving least-squares. Thus, this connection brings advantages from both approaches; on the one hand the matrix-free nature of approximate approximations and, on the other hand, the capacity of radial basis functions to interpolate functions from irregularly spaced points.

3.4.2 Connecting RBF and Iterated AMLS

Based on Equation 3.6, Fasshauer and Zhang [48] proposed the following iterative process, built upon approximate moving least-squares,

$$\begin{aligned} \mathcal{M}f^{(0)}(\mathbf{x}) &= \sum_{i=1}^{N} f_i \varphi_i(\mathbf{x}), & (3.8) \\ \mathcal{M}f^{(n+1)}(\mathbf{x}) &= \mathcal{M}f^{(n)}(\mathbf{x}) + \sum_{i=1}^{N} \left(f_i - \mathcal{M}f^{(n)}(\mathbf{x}_i)\right) \varphi_i(\mathbf{x}), & (3.9) \end{aligned}$$

and named it *iterated approximate moving least-squares approximation* (IAMLS). Fasshauer and Zhang proved, under easy-to-check conditions for the matrix $\mathbf{A} = \{\varphi_i(\mathbf{x}_j)\}_{i,j=1}^{N}$, that

$$\|\mathbf{A}\|_2 < 2 \Longrightarrow \mathcal{M}f^{(n)}(\mathbf{x}) \to \mathcal{M}f_{\mathcal{R}}(\mathbf{x}) \text{ when } n \to \infty,$$

where $\mathcal{M}f_{\mathcal{R}}(\mathbf{x})$ is the radial basis function interpolation for the basis $\{\varphi_i\}$. In fact, such a matricial condition is easily achieved, for instance, if $\{\varphi_i\}$ define a partition of unity, the matrix norm becomes $\|\mathbf{A}\|_2 = 1$. It is worth to mention that radial basis functions methods can suffer from numerical instability, for instance, for small ϵ when Gaussian basis functions are used. On the other hand, since the IAMLS produces a sequence of smooth solutions which converges to the radial basis functions solution, it produces more stable results as shown by Fasshauer and Zhang.

CHAPTER

4 MESHLESS SURFACES FROM POINT CLOUDS

Meshless surface rendering methods have been around for decades with the first work on point-based surface rendering proposed by Levoy and Whitted in 1985. However, it was not until some years ago that meshless modeling and rendering became popular. We refer the reader to surveys on the topic [133; 82]. Note that meshless methods are not limited to point-based techniques. In fact, a number of works on point-based rendering are based on a meshless surface representation defined by an implicit function or by a projection operator. Perhaps the ground-breaking work that most influenced this trend nowadays, is the work by Levin on *moving least-squares surfaces* [98]. At this point, it is necessary to remark that meshless surface approximation techniques can be used to generate a mesh as output. Nonetheless, the techniques remain meshless.

In this chapter, work developed upon recent results on moving least-squares surface approximation is reported. As seen before, moving least-squares is a powerful approximation method. Levin [97] studied the approximation power of the moving least-squares, setting the basis for a number of works that used the method to solve a large range of problems in several areas including computer graphics. The clear advantage of the moving least-squares method is that the accuracy of the approximation can be concentrated around a determined point in the domain, *e.g.*, the point where the function is evaluated. Although this means that an approximation has to be computed for every point where the function is evaluated, a very important consequence is that the data can be accurately approximated on the entire domain using polynomials of low degree locally.

The work described in Sections 4.2 and 4.3 was carried out in collaboration with João Paulo Gois from the Universidade de São Paulo, Brazil. Valdecir Polizelli-Junior and Tiago Etiene from the Universidade de São Paulo were active collaborators during the development of the techniques presented in Section 4.4 and must be credited for the implementation of the technique and development of the extensions to improve robustness. Different details of the hardware implementation of ray-tracing for projection operators (Section 4.5.1) were separately developed in collaboration with Tobias Schafhitzel from the Universität Stuttgart and Jõao Paulo Gois from the Universidade de São Paulo.

4.1 Meshless Surface Approximation

Before detailing the methods introduced here, an overview of previous work is given. The techniques presented in this thesis are to a great extent based on moving least-squares surfaces where, given a point set $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\} \subset \mathbb{R}^3$ sampled on a surface ∂S, we wish to find an approximation $\mathcal{M}\partial S$ to ∂S. As seen in Chapter 2, Levin defines $\mathcal{M}\partial S$ as the stationary points of a map f_{MLS}, given as a two-step procedure.

Amenta and Kil [11] generalized the moving least-squares surfaces and defined *extremal surfaces*. To that end, they start by giving an explicit definition of the moving least-squares surfaces as described in the following. Firstly, consider a vector field

$$\mathbf{n}(\mathbf{x}) = \operatorname*{argmin}_{\mathbf{a} \in \mathbb{S}^2} e_{MLS}(\mathbf{x}, \mathbf{a}),$$

where e_{MLS} is, as in Chapter 2, defined as

$$e_{MLS}(\mathbf{q}, \mathbf{a}) = \sum_{i=1}^{N} \left(\langle \mathbf{a}, \mathbf{x}_i \rangle - \langle \mathbf{a}, \mathbf{q} \rangle \right)^2 \omega_{MLS}(\mathbf{q}, \mathbf{x}_i).$$

This vector field can be usually uniquely determined since $\mathbf{x}$ is fixed and therefore e_{MLS} is a quadratic function of $\mathbf{a}$. As noted by Amenta and Kil, the set of points where $\mathbf{n}$ is not well-defined form surfaces which separate the space into regions, within each of which $\mathbf{n}$ is a smooth function of $\mathbf{x}$.

Then, given the line $l_{\mathbf{x},\mathbf{n}(\mathbf{x})}$ through $\mathbf{x}$ with direction $\mathbf{n}(\mathbf{x})$, Amenta and Kil characterize the moving least-squares surfaces as the set of points $\mathbf{x}$ for which $\mathbf{n}(\mathbf{x})$ is well-defined and $\mathbf{x}$ locally minimizes $e_{MLS}(\mathbf{y}, \mathbf{n}(\mathbf{x}))$, where $\mathbf{y} \in l_{\mathbf{x},\mathbf{n}(\mathbf{x})}$. The authors presented a proof that this characterization defines points on the moving least-squares surface and generalized it to define extremal surfaces by letting $\mathbf{n}$ be any function that assigns directions to points in space and using any function $e(\mathbf{x}, \mathbf{a})$ as energy function. Furthermore, if $\mathbf{n}$ is a consistently oriented smooth vector field, the implicit surface associated with an extremal surface defined by $\mathbf{n}$ and $e(\mathbf{x}, \mathbf{a})$ is given by

$$g(\mathbf{x}) = \langle \mathbf{n}(\mathbf{x}), \nabla_y e(\mathbf{y}, \mathbf{n}(\mathbf{x}))|_x \rangle = 0,$$

where $\nabla_y e(\mathbf{y}, \mathbf{n}(\mathbf{x}))|_x$ is the gradient of e as a function of y, when $\mathbf{n}(\mathbf{x})$ is fixed, evaluated at $\mathbf{x}$.

Implicit definitions using moving least-squares have been studied by other authors as well. Adamson and Alexa [2] presented a simple surface definition based on weighted averages and weighted covariances. Given a point $\mathbf{x}$, the weighted average is given by

$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \mathbf{x}_i \omega_{SMLS}(\mathbf{x}, \mathbf{x}_i)}{\sum_{i=1}^{N} \omega_{SMLS}(\mathbf{x}, \mathbf{x}_i)},$$

where

$$\omega_{SMLS}(\mathbf{x}, \mathbf{x}_i) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2/h^2),$$

and the weighted covariance at $\mathbf{x}$ in direction $\mathbf{n}$, describing how well a plane $\langle \mathbf{n}, \mathbf{x} - \mathbf{x}_i \rangle$ fits the weighted points, is given by

$$\sigma_n^2(\mathbf{x}) = \frac{\sum_{i=1}^{N} \langle \mathbf{n}, \mathbf{x} - \mathbf{x}_i \rangle^2 \omega_{SMLS}(\mathbf{x}, \mathbf{x}_i)}{\sum_{i=1}^{N} \omega_{SMLS}(\mathbf{x}, \mathbf{x}_i)}.$$

Let $\sigma(\mathbf{x})$ be the vector of weighted covariances along the directions of the canonical base, *i.e.*,

$$\sigma(\mathbf{x}) = \begin{pmatrix} \sigma_{(1,0,0)}(\mathbf{x}) \\ \sigma_{(0,1,0)}(\mathbf{x}) \\ \sigma_{(0,0,1)}(\mathbf{x}) \end{pmatrix}$$

then the directions of smallest and largest weighted covariances at $\mathbf{x}$ can be computed as the eigenvectors of the bilinear form

$$\sum(\mathbf{x}) = \sigma(\mathbf{x})\sigma(\mathbf{x})^T,$$

where an eigenvalue is the covariance along the direction of the associated eigenvector. Thus, the normal direction $\mathbf{n}(\mathbf{x})$ at $\mathbf{x}$ is given by the direction of smallest weighted covariance. The implicit definition of the surface in this case is given as the zero set of the function

$$I_{SMLS}(\mathbf{x}) = \langle \mathbf{n}(\mathbf{x}), \mathbf{a}(\mathbf{x}) - \mathbf{x} \rangle.$$

Kolluri [83] analyzed the implicit moving least-squares surface definition originally proposed by Shen *et al.* [141] to approximate polygon soups when used to approximate a point cloud. If ∂S is a smooth closed surface and each point $\mathbf{x}_i$ is equipped with an outside surface normal $\mathbf{n}_i$, the approximated surface $\mathcal{M}\partial S$ is defined as the zero set of the function

$$I_{IMLS}(\mathbf{x}) = \frac{\sum_{j=1}^{N} \omega_{IMLS}(\mathbf{x}, \mathbf{x}_i) \langle \mathbf{x} - \mathbf{x}_i, \mathbf{n}_i \rangle}{\sum_{j=1}^{N} \omega_{IMLS}(\mathbf{x}, \mathbf{x}_j)},$$

where

$$\omega_{IMLS}(\mathbf{x}, \mathbf{x}_i) = \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2/h^2)/A_i$$

and A_i is the number of samples inside a ball of radius h centered at $\mathbf{x}_i$. Kolluri proved that the function I_{IMLS} is a good approximation to the signed distance function to the surface ∂S and that $\mathcal{M}\partial S$ is geometrically close and homeomorphic to ∂S under the following sampling condition. Let the *local feature size* $\digamma(\mathbf{x})$ of a point $\mathbf{x} \in \partial S$ be the distance from $\mathbf{x}$ to the nearest point of the medial axis of ∂S. A point set $\mathcal{X}$ is an h-sample if the distance from any point $\mathbf{x} \in \partial S$ to its closest sample in $\mathcal{X}$ is less than $h\mathcal{F}(\mathbf{x})$. The proof of correctness presented by Kolluri requires a *uniform h-sampling*. The set $\mathcal{X}$ is a uniform h-sampling if, after having being scaled so that $\mathcal{F}(\mathbf{x}) > 1; \forall \mathbf{x} \in \partial S$, the distance from each point $\mathbf{x} \in \partial S$ to its closest sample is less than h. Also, for each sample $\mathbf{x}_i$, the distance to its closest surface point $\mathbf{p}$ should be less than h^2. Moreover, the angle between $\mathbf{n}_i$ of a sample $\mathbf{x}_i$ and the normal $\mathbf{n}_\mathbf{p}$ of the closest surface point to $\mathbf{x}_i$ should be less than h. Finally, let $\alpha_\mathbf{x}$ be the number of samples inside a ball of radius h centered at $\mathbf{x}$. Kolluri assumes that for each point $\mathbf{x}$, if $\alpha_\mathbf{x} > 0$, the number of samples inside the ball of radius $2h$ centered at $\mathbf{x}$ is at most $8\alpha_\mathbf{x}$.

Dey and Sun [40], following on the work of Kolluri, presented an implicit moving least-squares surface definition able to deal with adaptively sampled points. Thus, the sampling condition is similar to that of Kolluri with the exception of the uniform sampling density. The form of the implicit function in this case is the same as in the work by Kolluri with the exception of the weighting function. Concretely, the implicit function is defined as

$$I_{AMLS}(\mathbf{x}) = \frac{\sum_{j=1}^{N} \omega_{AMLS}(\mathbf{x}, \mathbf{x}_i)\langle \mathbf{x} - \mathbf{x}_i, \mathbf{n}_i \rangle}{\sum_{j=1}^{N} \omega_{AMLS}(\mathbf{x}, \mathbf{x}_j)},$$

where

$$\omega_{AMLS}(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{\sqrt{2}\|\mathbf{x} - \mathbf{x}_i\|^2}{\rho_e^2 f(\widetilde{\mathbf{x}}) f(\widetilde{\mathbf{x}}_i)} \right),$$

where $\widetilde{\mathbf{x}}$ and $\widetilde{\mathbf{x}}_i$ are the nearest points to $\mathbf{x}$ and $\mathbf{x}_i$ on ∂S respectively. The function f is a smooth function arbitrarily close to $\mathcal{F}$, *i.e.*,

$$|f(\mathbf{x}) - \mathcal{F}(\mathbf{x})| < \beta \mathcal{F}(\mathbf{x})$$

for arbitrarily small $\beta > 0$. This is done since $\mathcal{F}$ is not smooth everywhere on ∂S. The factor $\sqrt{2}$ in the exponent of the weighting function is introduced for convenience in the proofs of correctness presented by Dey and Sun.

Multi-level partition of unity implicits were proposed by Ohtake *et al.* [122]. To define the supports of the partition of unity, the domain is decomposed using an octree. The reconstructed surface mesh is then obtained from a regular grid (resampled from the octree) using Bloomenthal's polygonizer [21]. The implicit function is given by

$$I_{MPU}(\mathbf{x}) = \sum_{i=1}^{M} Q_i(\mathbf{x}) \phi_i(\mathbf{x}),$$

where $\{\phi_1, \cdots, \phi_M\}$ is as set of M non-negative functions with compact support such that

$$\sum_{i=1}^{M} \phi_i(\mathbf{x}) = 1,$$

$Q_i \in \mathcal{V}_i$, and $\mathcal{V}_i$ is a set of local approximation functions associated with each sub-domain $\mathrm{supp}(\phi_i)$. The set of functions $\{\phi_i\}$ can be generated by letting

$$\phi_i(\mathbf{x}) = \frac{\omega_{MPU}(\mathbf{c}_i, \mathbf{x})}{\sum_{i=1}^{M} \omega_{MPU}(\mathbf{c}_i, \mathbf{x})},$$

where the quadratic B-spline $b(t)$ is used to generate weight functions

$$\omega_{MPU}(\mathbf{c}_i, \mathbf{x}) = b\left(\frac{3\|\mathbf{x} - \mathbf{c}_i\|}{2R_i}\right)$$

centered at $\mathbf{c}_i$ and having a spherical support of radius R_i. An interpolation of $\mathcal{X}$ can be obtained by using

$$\omega_{MPU}(\mathbf{c}_i, \mathbf{x}) = \left[\frac{(R_i - \|\mathbf{x} - \mathbf{c}_i\|)_+}{R_i\|\mathbf{x} - \mathbf{c}_i\|}\right]^2,$$

where

$$(a)_+ = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

Given an octree partition of the space, which defines the centers $\mathbf{c}_i$ and radii R_i of the supports, the local fitting of the data is calculated using general (three-variate) quadrics or bivariate quadratic polynomials. In the latter case, a local coordinate system (η, ζ, χ) is defined for each support $(\mathbf{c}_i, R_i)$ with origin at $\mathbf{c}_i$, such that the plane (η, ζ) is orthogonal to $\mathbf{n}_i$ and the positive direction of χ coincides with the direction of $\mathbf{n}_i$, where $\mathbf{n}_i$ is the normal vector estimated at $\mathbf{c}_i$ using covariance analysis. Once the local system is constructed, the local polynomial approximation is defined as

$$Q_i(\mathbf{x}) = \chi - \left(c_1\eta^2 + 2c_2\eta\zeta + c_3\zeta + c_4\eta + c_5\zeta + c_6\right).$$

The unknown coefficients c_k; $k = 1, \cdots, 6$, are determined by minimizing

$$\sum_{j=1}^{N} Q_i(\mathbf{x}_j)^2 \omega_{MPU}(\mathbf{c}_i, \mathbf{x}_j).$$

Ohtake *et al.* [123] later extended their work to improve the resulting surface by using normalized radial basis functions. Basically, the implicit function is defined as

$$I_{RMPU} = \sum_{i=1}^{M} Q_i(\mathbf{x})\phi_i(\mathbf{x}) + \sum_{i=1}^{M} \gamma_i\phi_i(\mathbf{x}),$$

where $\{\phi_i\}$ is a set of normalized radial basis functions, and γ_i; $i = 1, \cdots, M$, are the coefficients of the radial basis functions interpolation to be determined.

Kazhdan *et al.* [79] proposed an interesting approach in which the surface approximation is formulated as a Poisson problem. This method presents several advantages over other formulations, but its processing time is higher than the one needed by partition of unity and moving least-squares formulations. This technique is based on the insight that there is an integral relationship between oriented points sampled from the surface of a model and the *indicator function* of the model. The indicator function χ_S for a model S is defined as 1 at the points inside S, and 0 outside S. Given a patch $\mathcal{P}_{\mathbf{x}_i} \subset \partial S$ for each $\mathbf{x}_i$, the authors start by constructing a gradient field

$$\mathbf{V}(\mathbf{x}) \equiv \sum_{i=1}^{N} |\mathcal{P}_{\mathbf{x}_i}| \widetilde{F}_{\mathbf{x}_i}(\mathbf{x}) \mathbf{n}_i,$$

where $\widetilde{F}_{\mathbf{x}_i}(\mathbf{x}) = \widetilde{F}(\mathbf{x} - \mathbf{x}_i)$, $\widetilde{F}$ is a smoothing filter, and $|\mathcal{P}_{\mathbf{x}_i}|$ is the area of the patch $\mathcal{P}_{\mathbf{x}_i}$. This vector field approximates the gradient field of the filtered indicator function, *i.e.*,

$$\nabla(\widetilde{\chi}) = \nabla(\chi_S * \widetilde{F}) \approx \mathbf{V},$$

where $(\cdot * \cdot)$ is the convolution operator. This problem can be regarded as a Poisson problem, for which the solution of the Poisson equation

$$\triangle \widetilde{\chi} = \nabla \cdot \mathbf{V}$$

provides the best least-squares approximation.

Lipman *et al.* [104] presented an approximation technique based on moving least-squares able to faithfully reconstruct piecewise-smooth surfaces from unorganized point sets. The method finds, for each projected point, a proper local approximation space of piecewise polynomials. The locally constructed spline encapsulates the local singularities which may exist in the data, which constitutes a very important contribution for the meshless surface reconstruction community. This technique will be described in detail later in this chapter in the context of the work on approximate moving least-squares surfaces presented here.

Lipman *et al.* [105] also developed a projection operator for surface reconstruction, which is parameterization free, in the sense that is does not rely on estimating a local parametric representation, such as the local tangent planes used by most of the approaches described in this section. Given the input point set $\mathcal{X} = \{\mathbf{x}_i : i = 1, \cdots, N\} \subset \mathbb{R}^3$, the operator maps an arbitrary point set $\mathcal{P}^{[0]} = \left\{\mathbf{p}_j^{[0]} : j = 1, \cdots, M\right\} \subset \mathbb{R}^3$ onto the set $\mathcal{X}$. The goal is to compute a set of projected points $\mathcal{Q} = \{\mathbf{q}_j : j = 1, \cdots, M\} \subset \mathbb{R}^3$ such that it minimizes the sum of weighted distances to points of $\mathcal{X}$, with respect to radial weights centered at the same set of points $\mathcal{Q}$. Furthermore, the points in $\mathcal{Q}$ should not be too close to each other. Thus, the desired set of points $\mathcal{Q}$ is defined as the fixed point solution of the equation

$$\mathcal{Q} = G(\mathcal{Q}),$$

where

$$\begin{aligned}
G(\mathcal{C}) &= \underset{\mathcal{P}=\{\mathbf{p}_j:j=1,\cdots,M\}}{\operatorname{argmin}} \{E_1(\mathcal{P},\mathcal{X},\mathcal{C}) + E_2(\mathcal{P},\mathcal{C})\},\\
E_1(\mathcal{P},\mathcal{X},\mathcal{C}) &= \sum_{i=1}^{N}\sum_{j=1}^{M} \|\mathbf{p}_j - \mathbf{x}_i\|\omega_{LOP}(\mathbf{c}_j,\mathbf{x}_i),\\
E_2(\mathcal{P},\mathcal{C}) &= \sum_{k=1}^{M}\lambda_k \sum_{j=1,j\neq k}^{M} \eta(\|\mathbf{p}_k - \mathbf{c}_j\|)\omega_{LOP}(\mathbf{c}_j,\mathbf{c}_k),
\end{aligned}$$

where $\omega_{LOP}(\mathbf{p},\mathbf{q}) \equiv w_{LOP}(\|\mathbf{p}-\mathbf{q}\|)$, and w_{LOP}, as before, is a fast-decreasing smooth weight function with compact support radius h defining the size of the influence radius, $\eta(r)$ is another decreasing function penalizing $\mathbf{p}_k$ which become too close to other points, and $\Lambda = \{\lambda_k : k = 1,\cdots,M\}$ is a set of balancing terms. Intuitively, the term E_1 drives the projected points in $\mathcal{Q}$ to approximate the geometry of $\mathcal{X}$, and the term E_2 strives at keeping the distribution of the points in $\mathcal{Q}$ fair. The authors prove the approximation order of the operator and provide a means to compute suitable values for λ_k.

Guennebaud and Gross [62] presented a surface approximation method based on moving least-squares fitting of algebraic spheres. The use of the algebraic sphere to locally approximate the data improves stability under low sample rates and in the presence of high curvature. An algebraic sphere is defined as the zero set of the scalar field $s_{\mathbf{u}}(\mathbf{x}) = [1, \mathbf{x}^T, \mathbf{x}^T\mathbf{x}]\mathbf{u}$, where $\mathbf{u} = [u_0,\cdots,u_{s+1}]^T \in \mathbb{R}^{s+2}$ is the vector of coefficients describing the sphere, with s being the dimension. In degenerate cases, u corresponds to the coefficients of a plane with u_0 representing the distance from the origin, $[u_1,\cdots,u_s]^T$ being its normal, and $u_{s+1} = 0$. To fit the algebraic sphere to a set of N points, let $\mathbf{W}(\mathbf{x})$ and $\mathbf{D}$ respectively be the $N \times N$ diagonal weight matrix and the $N \times (s+2)$ design matrix defined as

$$\mathbf{W}(\mathbf{x}) = [\omega_{APSS}(\mathbf{x}_1,\mathbf{x}),\cdots,\omega_{APSS}(\mathbf{x}_N,\mathbf{x})],\ \mathbf{D} = \begin{bmatrix} 1 & \mathbf{x}_1^T & \mathbf{x}_1^T\mathbf{x}_1 \\ . & . & . \\ . & . & . \\ . & . & . \\ 1 & \mathbf{x}_N^T & \mathbf{x}_N^T\mathbf{x}_N \end{bmatrix},$$

where

$$\omega_{APPS}(\mathbf{x}_i,\mathbf{x}) = \phi\left(\frac{\|\mathbf{x}_i - \mathbf{x}\|}{h}\right)$$

and

$$\phi(r) = \begin{cases} (1-r^2)^4 & r < 1 \\ 0 & \text{otherwise.} \end{cases}$$

Then, the solution $\mathbf{u}(\mathbf{x})$ can be expressed as

$$\mathbf{u}(\mathbf{x}) = \underset{\mathbf{u},\mathbf{u}\neq 0}{\operatorname{argmin}} \left\| \mathbf{W}^{\frac{1}{2}}(\mathbf{x})\mathbf{D}\mathbf{u} \right\|^2 . \tag{4.1}$$

In order to avoid the solution $\mathbf{u}(\mathbf{x}) = 0$, Pratt's constraint is used where the norm of the gradient at the surface of the sphere is constrained to unit length by means of a quadratic normalization, namely, $\|(u_1, \cdots, u_s)\|^2 - 4u_0u_{s+1} = 1$. This ensures that the algebraically fitted sphere is close to the least-squares Euclidean best fit. This minimization is only used to estimate the normal vectors $\mathbf{n}_i$ at the points in $\mathcal{X}$. Once the normals are available, as input or by means of the fitting described above, the actual sphere fitting to reconstruct the surface is performed. To that end, the constraints $\nabla s_{\mathbf{u}}(\mathbf{x}_i) = \mathbf{n}_i$ are added to the minimization problem of Equation 4.1. Therefore, the implicit function is defined as

$$I_{APSS} = s_{\mathbf{u}(\mathbf{x})}(\mathbf{x}) = \left[1, \mathbf{x}^T, \mathbf{x}^T\mathbf{x}\right] \mathbf{u}(\mathbf{x}).$$

This method is able to provide curvature information as a by-product of the fitting process. We, on the other hand, first estimate the curvature at a given point on the surface and exploit it to perform polynomial fitting using non-complete quadratic polynomials instead of full polynomials, as described in the following.

4.2 Curvature-driven Projection Operator

In this section, a novel projection operator for surface approximation from unorganized point sets is presented, based on the approximation of directional curvatures and the diffusion equation [59]. The anisotropic diffusion equation used helps preserve the geometry of the original surface and makes it possible to represent thin features in the model. Also, it is shown how principal curvatures and directions can be estimated for point clouds. This curvature information may be used to defined a local polynomial approximation as will be also described. The fact that the local approximation performed by the method proposed is defined as a non-complete quadratic polynomial can help decrease the processing time of algorithms that perform intensive intersection computations such as ray-tracing. The ray-tracing algorithm by Adamson and Alexa [3] is used to render the surfaces defined by the curvature-driven projection operator. To that end, the intersection computation must be modified to fit the proposed projection operator. This ray-tracer is used to demonstrate the quality of the approximations obtained (see Figure 4.1 for an example).

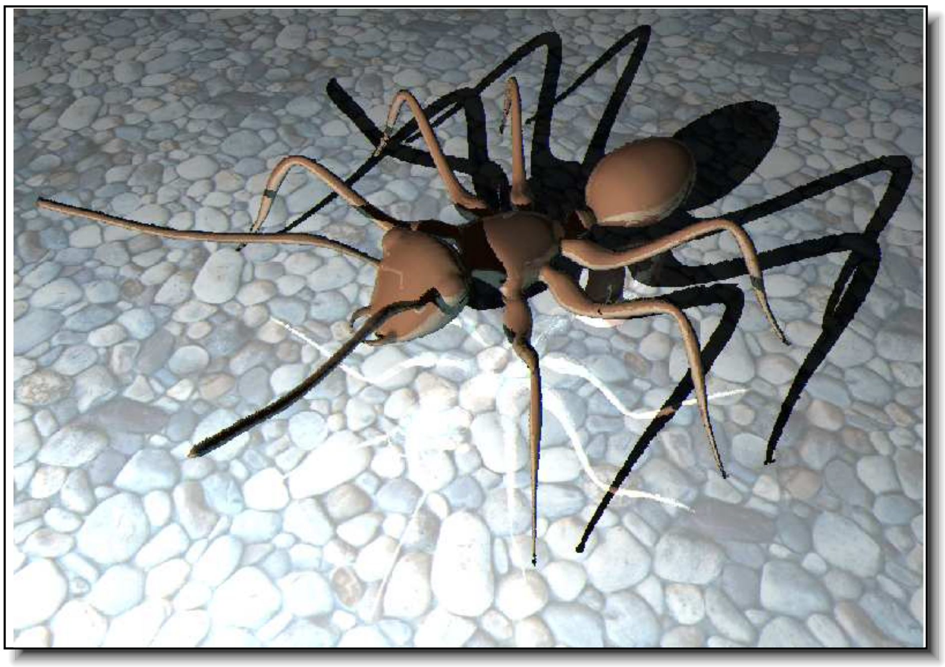


Figure 4.1: Rendering of the approximate surface for the EtiAnt dataset obtained with the curvature-driven projection operator described in this section (see color plates).

Lange and Polthier [90] adapted the well known mesh-oriented method by Taubin [148] for surface fairing to the point cloud context. In their work, the authors used the method by Taubin together with an anisotropic diffusion equation for removing noise from point clouds without smoothing sharp corners.

Curvatures and principal directions estimation for regular grids and polygonal meshes has been extensively studied in both the qualitative and the quantitative cases. Maltred and Daniel [108] presented a survey on classical work on curvature estimation and a classification based on the requirements and constraints of the methods described. Although there are many methods for estimating principal curvatures and directions, almost all of them need a mesh. Only recently, effective methods to estimate curvature information directly from point sets were presented [4; 154; 90; 72].

Tong and Tang [154] presented a robust curvature estimation method based on adaptive curvature tensors by means of tensor voting. In addition, they presented an analytical comparison with classical and efficient methods with respect to their input (point clouds or mesh models), their requirements (geometrical measures) and their outputs (quantitative or qualitative estimations). Huang and Menq [72] proposed a curvature estimation method built upon the least-squares scheme and Euler's theorem from differential geometry. Although the method was proposed to locally optimize unstructured surface meshes, it is also suitable for point clouds.

We derive the method presented from this work, since it can be easily adapted to support weighting functions, which is an important condition for the quality of the models generated by meshless techniques. Thus, the curvature estimation method by Huang and Menq was modified to introduce weighting functions in order to improve the robustness of the method. These weighting functions were carefully constructed and are specific to the problem at hand.

4.2.1 Principal directions and curvatures

The directional curvatures at a point $\mathbf{x}$ on a smooth surface ∂S are defined in terms of the curvatures of smooth curves containing $\mathbf{x}$. The minimum and maximum directional curvatures computed from the curves are called *principal curvatures* and their respective directions are called *principal directions*. One important result is that principal directions are orthogonal at $\mathbf{x}$ [28].

Euler's theorem from differential geometry states that every directional curvature in $\mathbf{x} \in \partial S$ can be described as a function of its principal directions and curvatures. More formally, let us define the principal directions and curvatures at $\mathbf{x}$ as $\nu_1^{\mathbf{x}}$ and $\nu_2^{\mathbf{x}}$, and $\kappa_1^{\mathbf{x}}$ and $\kappa_2^{\mathbf{x}}$ respectively. Euler's theorem states that the curvature at $\mathbf{x}$ in the direction μ is given by:

$$\kappa^{\mathbf{x}}(\mu) = \kappa_1^{\mathbf{x}} \cos^2(\alpha) + \kappa_2^{\mathbf{x}} \sin^2(\alpha), \tag{4.2}$$

where $\mu = \cos(\alpha)\nu_1^{\mathbf{x}} + \sin(\alpha)\nu_2^{\mathbf{x}}$.

Let us consider the approximated tangent plane H at $\mathbf{x}$, where a local orthonormal coordinate system (η, ζ, χ) with origin at $\mathbf{x}$ is defined, such that the plane (η, ζ) is parallel to H (Figure 4.2). The projections $\tilde{\mathbf{x}}_i$ on H of the points $\mathbf{x}_i \in \mathcal{X}$ are computed, which define the directions ν_i. In practice only a subset of $\mathcal{X}$ consisting of the nearest neighbors of $\mathbf{x}$ are used. However, as before, this restriction is not included in the formulation here, since later by introducing weighting functions, it is ensured that only the meaningful points in $\mathcal{X}$ for the computations at $\mathbf{x}$ are taken into account.

By sorting such directions in counterclock direction, the angles θ_i from η to $\tilde{\mathbf{x}}_i$ are obtained. Denote by β_i the counterclockwise angle from the principal direction $\nu_1^{\mathbf{x}}$ to ν_i. The directional curvature can be approximated by

$$\kappa^{\mathbf{x}}(\nu_i) := \frac{2\langle \mathbf{n}(\mathbf{x}), \varrho^i \rangle}{\langle \varrho^i, \varrho^i \rangle}, \tag{4.3}$$

where $\mathbf{n}(\mathbf{x})$ is the normal vector at $\mathbf{x}$ and $\varrho^i = \mathbf{x}_i - \mathbf{x}$ for $\mathbf{x}_i$. The following nonlinear overdetermined system is obtained from Euler's theorem and the directional curvatures approximation (Equation 4.3)

$$\left\{\kappa_1^{\mathbf{x}} \cos^2(\beta_1 + \delta\beta_i) + \kappa_2^{\mathbf{x}} \sin^2(\beta_1 + \delta\beta_i) = \kappa^{\mathbf{x}}(\nu_i);\ i = 1, \ldots, N\right. , \tag{4.4}$$

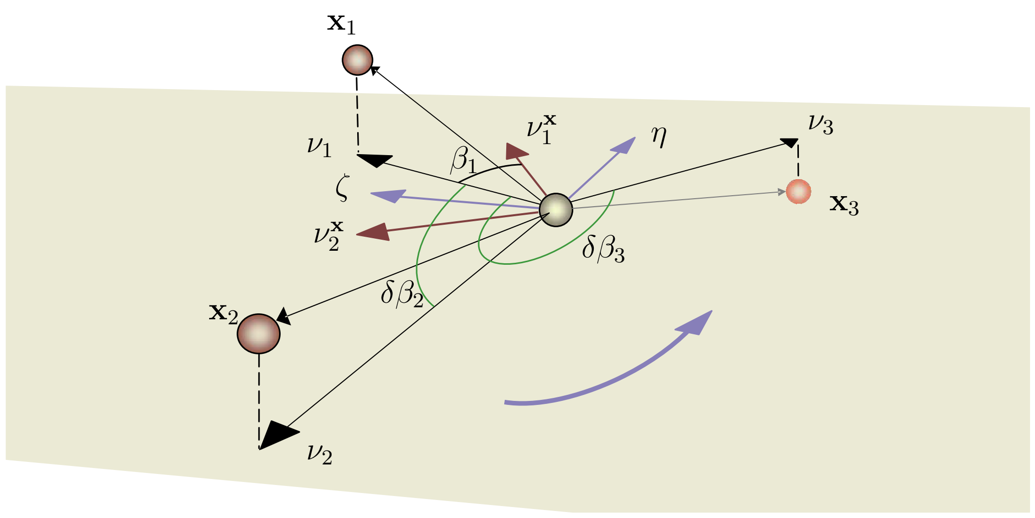


Figure 4.2: Estimating directional curvatures on the approximated tangent plane at $\mathbf{x}$ (see color plates).

where $\delta\beta_i = \beta_i - \beta_1 = \theta_i - \theta_1$. Note that $\delta\beta_1 = 0$. Following the work by Huang and Menq, let us define

$$\gamma_1 = \frac{1}{2}(\kappa_1^{\mathbf{x}} + \kappa_2^{\mathbf{x}}), \quad \gamma_2 = -\cos(2\beta_1)(\kappa_2^{\mathbf{x}} - \kappa_1^{\mathbf{x}}), \quad \gamma_3 = \sin(2\beta_1)(\kappa_2^{\mathbf{x}} - \kappa_1^{\mathbf{x}}),$$

to obtain the following overdetermined linear system

$$\{\gamma_1 + \gamma_2\cos(2\delta\beta_i) + \gamma_3\sin(2\delta\beta_i) = \kappa^{\mathbf{x}}(\nu_i);\ i = 1,\ldots,N\,. \tag{4.5}$$

Therefore, the normal equation for System 4.5 becomes

$$\begin{bmatrix} N & \sum_{i=1}^{N} c_i & \sum_{i=1}^{N} s_i \\ \sum_{i=1}^{N} c_i & \sum_{i=1}^{N} c_i^2 & \sum_{i=1}^{N} c_i s_i \\ \sum_{i=1}^{N} s_i & \sum_{i=1}^{N} c_i s_i & \sum_{i=1}^{N} s_i^2 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} \hat{\kappa}_i \\ \sum_{i=1}^{N} \hat{\kappa}_i c_i \\ \sum_{i=1}^{N} \hat{\kappa}_i s_i \end{bmatrix}, \tag{4.6}$$

where $c_i = \cos(2\delta\beta_i)$, $s_i = \sin(2\delta\beta_i)$ and $\hat{\kappa}_i = \kappa^{\mathbf{x}}(\nu_i)$. Thus, the principal directions and curvatures are straightforwardly obtained from γ_i, $1 \leq i \leq 3$.

Although Huang and Menq state that the method is robust for noisy data, robustness for the problem at hand was only achieved by adding suitable weights.

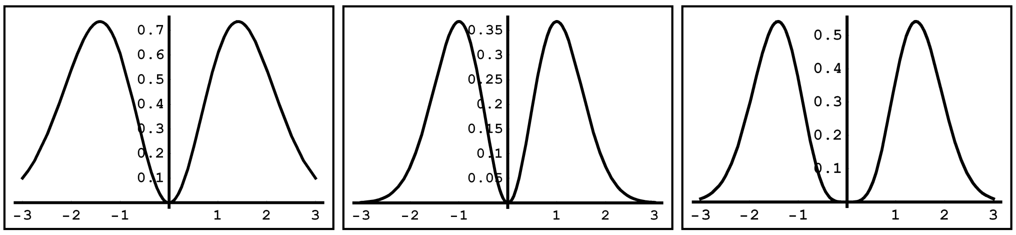


Figure 4.3: Examples of "M"-like function graphs. From left to right: $h = 2$ and $\varsigma = 1$, $h = 1$ and $\varsigma = 1$, and $h = 1$ and $\varsigma = 2$.

To that end, the following weighting function was introduced for the first time in the process

$$\omega_{CPO}(\mathbf{x}_i, \mathbf{x}) = \|\mathbf{x} - \mathbf{x}_i\|^{2\varsigma} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h^2}\right), \tag{4.7}$$

where $\varsigma \in \mathbb{N}^*$. Note that this function is an "M"-like function. Thus, by letting $\omega_i \equiv \omega_{CPO}(\mathbf{x}_i, \mathbf{x})$, the normal equation becomes

$$\begin{bmatrix} \sum_{i=1}^{N} \omega_i & \sum_{i=1}^{N} \omega_i c_i & \sum_{i=1}^{N} \omega_i s_i \\ \sum_{i=1}^{N} \omega_i c_i & \sum_{i=1}^{N} \omega_i c_i^2 & \sum_{i=1}^{N} \omega_i c_i s_i \\ \sum_{i=1}^{N} \omega_i s_i & \sum_{i=1}^{N} \omega_i c_i s_i & \sum_{i=1}^{N} \omega_i s_i^2 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} \omega_i \hat{\kappa}_i \\ \sum_{i=1}^{N} \omega_i \hat{\kappa}_i c_i \\ \sum_{i=1}^{N} \omega_i \hat{\kappa}_i s_i \end{bmatrix}. \tag{4.8}$$

The use of an "M"-like function is very important to obtain good results with the method proposed. This is due to the fact that the directional curvatures obtained by Equation 4.3 are dependent on the position of the points and their normal vectors. A small perturbation in the position of point $\mathbf{x}$ can produce a quite different solution with the original method by Huang.

In the "M"-like function, $\|\mathbf{x} - \mathbf{x}_i\|^{2\varsigma}$ controls the influence of the points close to $\mathbf{x}$ on the solution. The larger ς is, the smaller the region around $\mathbf{x}$ which will have significant influence in the solution is. The exponential member of the function maintains the same behavior of the traditional Gaussian function. Examples of graphs of this function are presented in Figure 4.3 with different parameter values.

Unlike a Gaussian weight, the "M"-like function is able to solve the problem and to increase robustness. This will be shown in the next section after the description of the surface approximation procedure.

4.2.2 Projection and rendering procedures

As mentioned before, traditional projection-based surface reconstruction techniques define the approximated surface for a given point cloud as the set of stationary points for a carefully designed projection operator. This way, the input point cloud can be resampled by projecting a sufficiently large number of points from its neighborhood onto the approximated surface. With this procedure, a dense sampling that covers the image space consistently can be obtained.

In this section, a new projection operator, derived from the diffusion equation and directional curvature information, is described. Also, a local approximation to the surface, that is not part of the projection procedure, is obtained using the method for computing principal curvatures and directions given in the previous section and a result from differential geometry, which states that a surface can be approximated locally (in the neighborhood of a point $\mathbf{x} \in \partial\mathcal{S}$) by

$$\mathcal{M}f_{\kappa}(\xi,\eta) = \frac{1}{2}\left(\kappa_1^{\mathbf{x}}\xi^2 + \kappa_2^{\mathbf{x}}\eta^2\right), \tag{4.9}$$

where (ξ,η) is in the local coordinate system defined by the principal directions at $\mathbf{x}$.

The projection procedure is based on the surface fairing method proposed by Lange and Polthier [90] for point clouds. The authors make use of an anisotropic diffusion equation, which is useful to preserve sharp corners. Let us consider the diffusion equation

$$\frac{\partial \mathbf{x}}{\partial t} = \lambda \Delta \mathbf{x}, \tag{4.10}$$

where $\Delta\mathbf{x}$ is the Laplacian of $\mathbf{x}$ and λ is the diffusive term. The Laplace operator can be approximated by the umbrella operator

$$\tilde{\Delta}\mathbf{x} = \frac{1}{\Omega}\sum_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})} \omega_i \cdot (\mathbf{x}_i - \mathbf{x}), \tag{4.11}$$

where $\mathcal{N}(\mathbf{x})$ is the set of k nearest neighbors of $\mathbf{x}$, k being user-defined,

$$\omega_i = \frac{1}{\|\mathbf{x}_i - \mathbf{x}\|^2}$$

and $\Omega = \sum \omega_i$. The explicit forward Euler method for Equation 4.10 leads to

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \lambda\delta t\tilde{\Delta}\mathbf{x}^{(n)}, \tag{4.12}$$

where δt is the step size and $\mathbf{x}^{(n)}$ is the integration at iteration n. It must be observed that $\lambda\delta t$ must satisfy the time step conditions [68]. Lange and Polthier modified the traditional umbrella operator to obtain an anisotropic operator by introducing a suitable real function which offers information related to the shape of the object. This operator is able to move a point onto the surface fairly. The anisotropic Laplacian becomes

$$\tilde{\Delta}_{\Lambda}\mathbf{x} = \frac{1}{\Omega}\sum_{\mathbf{x}_i \in \mathcal{N}(\mathbf{x})} \Lambda_i \cdot (\mathbf{x}_i - \mathbf{x}), \tag{4.13}$$

where Λ_i is a real function which depends on the directional curvatures (Equation 4.3) at $\mathbf{x}$. Lange and Polthier argued for the use of one of the following functions for a given threshold ε

$$\Lambda_i = \begin{cases} 1 & |\kappa^{\mathbf{x}}(\nu_i)| < \varepsilon \\ 0 & \text{otherwise;} \end{cases} \tag{4.14}$$

$$\Lambda_i = \begin{cases} 1 & |\kappa^{\mathbf{x}}(\nu_i)| < \varepsilon \\ \frac{\lambda^2}{\lambda^2 + 10(|\kappa^{\mathbf{x}}(\nu_i)| - \lambda)^2} & \text{otherwise.} \end{cases} \tag{4.15}$$

With this framework, the projection $\mathbf{p}$ of a point $\mathbf{r}$ onto the surface and the local approximation to the surface at $\mathbf{p}$ can be found using following procedure:

```
1 Find the plane $H$ with normal vector $\mathbf{l} = \frac{1}{\Omega}\sum \omega_i \cdot \mathbf{n}_i$ passing through
  $\mathbf{o} = \frac{1}{\Omega}\sum \omega_i \cdot \mathbf{x}_i$, where $\mathbf{n}_i$ is the normal vector at $\mathbf{x}_i$;
2 Find the projection $\mathbf{q}$ of $\mathbf{r}$ on $H$;
3 Find the projected point $\mathbf{p}$ resulting from using the anisotropic diffusion
  equation starting at $\mathbf{q}$;
4 Calculate the principal directions $\nu_1^{\mathbf{p}}, \nu_2^{\mathbf{p}}$ and curvatures $\kappa_1^{\mathbf{p}}, \kappa_2^{\mathbf{p}}$ at $\mathbf{p}$;
5 Define a local coordinate system with axis $(\nu_1^{\mathbf{p}}, \nu_2^{\mathbf{p}})$ and origin at $\mathbf{p}$;
6 The polynomial $\frac{1}{2}(\kappa_1^{\mathbf{p}}\xi^2 + \kappa_2^{\mathbf{p}}\eta^2)$, where $\xi$ and $\eta$ are in the local coordinate
  system, approximates the surface locally.
```

The approximated surface is defined as the set of stationary points for the projection process described above. To render the approximated surface, the ray-tracing algorithm proposed by Adamson and Alexa [3] is used, changing only the projection procedure. This can be done because the process above described provides not only the projection of the point but also a local polynomial approximation to the surface (see Section 2.2.3). However, a good estimate of the principal curvatures ($\kappa_1^{\mathbf{p}}$, $\kappa_2^{\mathbf{p}}$) and directions ($\nu_1^{\mathbf{p}}$, $\nu_2^{\mathbf{p}}$) is of major importance to obtain a good local approximation to the surface (Equation 4.9). As claimed in the previous section, only by introducing suitable weights into the curvature estimation a robust approximation can be obtained. Figure 4.4 shows the effect of introducing weights into System 4.8. As can be seen, a Gaussian weight does not solve the problem. The use of the "M"-like function was inspired by the fact that instabilities may arise in the estimate of the directional curvature given by Equation 4.3 when $\mathbf{x} \to \mathbf{x}_i$ for some $\mathbf{x}_i \in \mathcal{X}$. Note, however, that the "M"-like function is used only for the curvature estimation, while a Gaussian weight is used for the rest of the process.

The moving least-squares-based surface approximation method proposed by Alexa *et al.* [5] was implemented to compare the results obtained with their method and the method presented here. The goal was to be able to obtain a comparable approximation to the surface using a reduced polynomial obtained from the curvature information. Therefore, complete quadratic polynomials were used in the

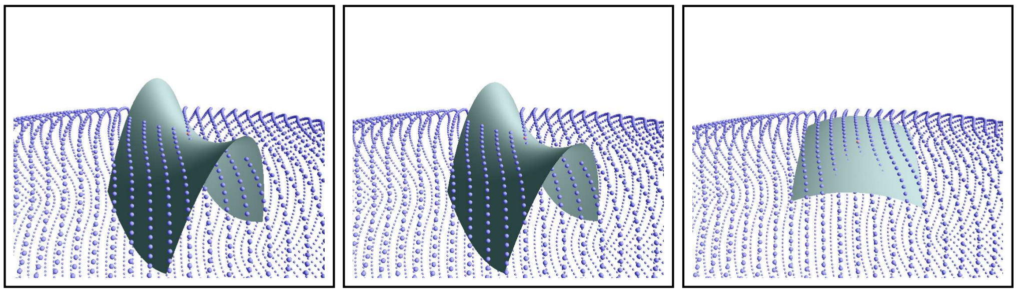


Figure 4.4: Polynomial approximations obtained (from left to right) without weights, with Gaussian weights and with the “M”-like function (top row).

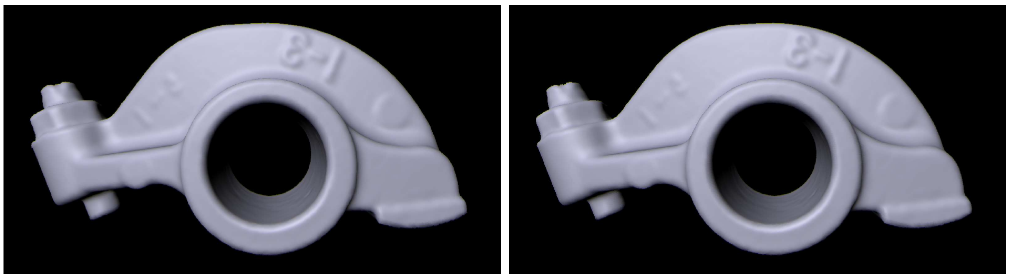


Figure 4.5: Ray-tracing of the approximated surface for the Rocker Arm dataset obtained with the moving least-squares-based (left) and the curvature-driven (right) methods.

implementation of Alexa’s method. In Figure 4.5 the results of both the moving least-squares-based approximation and the curvature-driven approximation for the Rocker Arm dataset are shown. In both cases, the ray-tracing algorithm by Adamson and Alexa was used to render the approximated surface. As can be seen in the figure, the results are equally good, however the method presented is between 1.5 to 2.5 times faster. For these performance tests, the models were rendered with no reflection or refraction effects to a 533×400 viewport. More complex scenes were also rendered and are shown in Figures 4.6 and 4.1. Note that thin features in the EtiAnt model in Figure 4.1 were correctly reconstructed with our method.

The difference in the processing time is due to the fact that, although the method proposed makes use of trigonometric functions to construct the normal equation, it only needs to solve a linear system with three unknowns and thus closed formulations can be used. Although a complete quadratic polynomial can better approximate a larger neighborhood than the non-complete quadratic polynomial used, for the local computations involved in the method proposed this latter quadratic polynomial approximates the surface accurately. Another important advantage of the method developed is the availability of an explicit characterization of the surface by means of the curvature information. Also, in practical terms, when computing the local approximation, the points for which the “M”-function

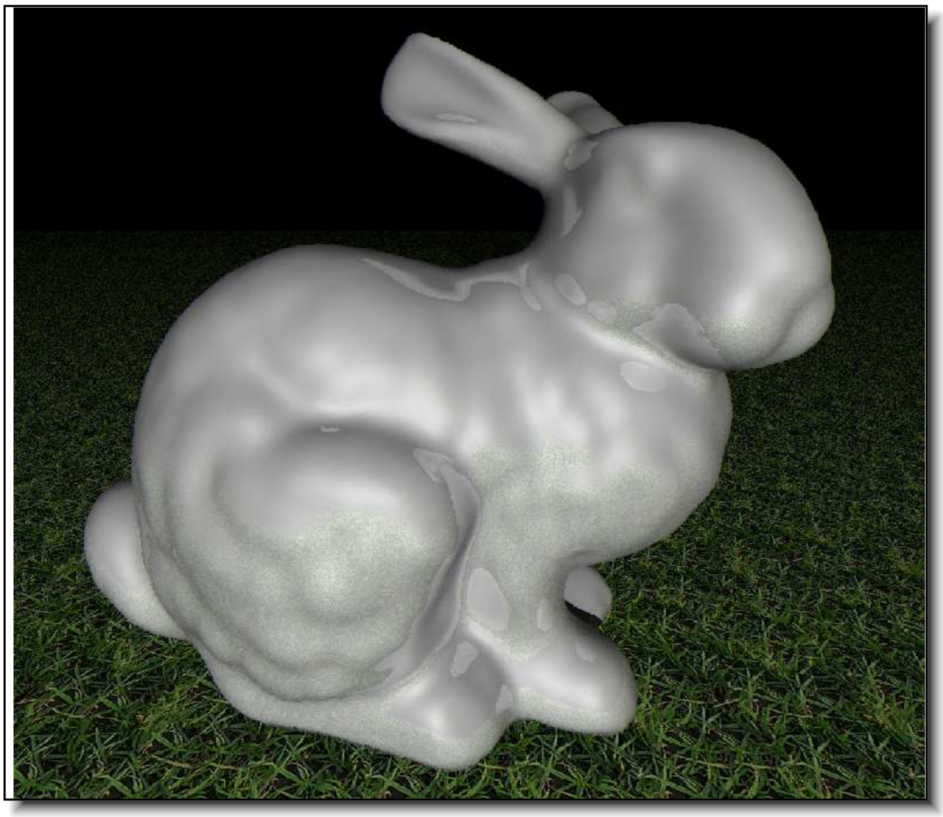


Figure 4.6: Rendering of the approximate surface for the Stanford Bunny dataset obtained with the curvature-driven projection operator described in this section.

has values lower than a threshold are discarded. This minimizes the computational cost since less trigonometrical operations are performed. Additionally, it reduces the possibility of numerical instability during the local approximation computation.

As mentioned before, the use of a non-complete quadratic polynomial for the polynomial approximations simplified the ray-surface intersection computation. However, it would be desirable to analyze the performance impact of the principal directions and curvature estimation process and of the anisotropic diffuse equation (although it was found that this latter process needs few iterations to converge). Acceleration techniques can be introduced into the implementation of the ray-tracing algorithm in order to be able to exploit the simplicity of the intersection computation.

Besides the potential gain in computation time, the availability of curvature information is important for a number of applications. This is a clear advantage of the method presented over other surface approximation techniques. There are few works in the literature for curvature estimation from clouds of points and so far there is no work presenting comparisons among these methods. A mathematical and computational efficiency study of such methods must be performed. Although, with the method proposed, local characterization of the surface offered by the principal directions and curvatures is available, it could be of interest to define global properties, *e.g.*, the Euler characteristic.

4.3 Approximate MLS Surfaces

In this section, two problems of approximation techniques based on moving least-squares are addressed. The first one is the need for solving a large number of small systems of equations when local polynomials are used to approximate the surface. This computational effort can be high for large models and viewport resolutions, turning the approaches prohibitively slow. On the other hand, methods that define the surface as the zero set of an implicit function are considerably faster. However, the implicit formulations presented in previous work provide a planar fitting to the surface, which is not able to model details in the data as well as methods based on local polynomial approximations of higher order. Although the work by Guennebaud and Gross [62] is an exception in that the implicit definition is based on local spheres fitting, a more general polynomial fitting might be desirable. Therefore, the goal is to define a surface approximation that combines the approximation power of local polynomial approximations obtained with moving least-squares and the simplicity and performance of the implicit formulations. To that end, a method free of systems of equations is proposed using recent results on approximate moving least-squares approximation. Connections between the theories on radial basis functions and on approximate moving least-squares approximation enable the use of such functions for meshless surface modeling and rendering of irregularly sampled point sets. By performing an iterative correction process, an approximation to the surface that better fits the sample points is obtained. The iterative process defined with iterated approximate moving least-squares generates a family of implicit surfaces ranging from the approximated moving least-squares implicit surface to a radial basis functions interpolated solution. The method is able to deal with noisy point sets by computing an estimate of the optimal number of iterations of this correction process. An example of a surface reconstructed with this method is shown in Figure 4.7.

The second problem addressed is the modeling of sharp features, *i.e.*, discontinuities in the approximating function or its derivatives. Point-based modeling and rendering techniques are, in general, not able to automatically represent sharp edges. This problem is depicted in Figure 4.8, where renderings of the Cube and Fan Disk datasets obtained with known point-based methods are shown. As can be seen, the surface approximation based on local polynomial approximations produces good results, but the edges are still smooth. This issue has been addressed

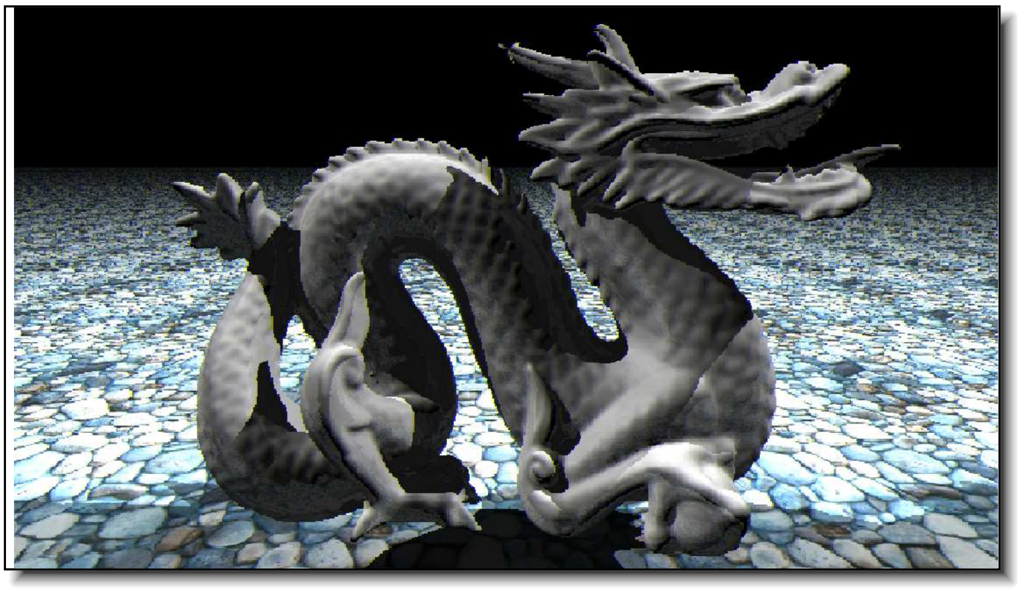


Figure 4.7: Approximated surface for the Stanford Dragon obtained as the zero set of the implicit function based on approximate moving least-squares approximation (see color plates).

by point-based methods so far using extensive tests, complex and computationally expensive statistical tools, auxiliary meshes, piecewise approximations and user intervention [104; 51; 130].

Fleishman *et al.* [51] make use of boolean operations where sharp edges are modeled by identifying them through detection of outliers. The method by Fleishman *et al.* uses a robust statistical technique, called forward search [14], to define piecewise local approximations. Given a point $\mathbf{x}$ to be projected onto the surface, the method computes a set of piecewise smooth surfaces in the neighborhood of $\mathbf{x}$. This is done by an incremental statistical method that adds points progressively to the local approximation, investigating the quality of the approximation each time a new point is added. Once such set of piecewise smooth surfaces is defined, the method projects the point according to possible neighborhood situations. Other methods making use of boolean operations have been previously addressed [63; 1; 124; 172]. To the extent of our knowledge, only the work by Reuter *et al.* [130] and by Lipman *et al.* [104] do not make use of boolean operators among point sets to define sharp edges.

The method by Reuter *et al.* makes use of the *enriched reproducing kernel particle approximation* to substitute the local polynomial approximation in the operator defined by Alexa *et al.* [5] for a local approximation that considers not only polynomial terms, but also enriched functions that describe, in some sense, discontinuous derivatives. The main drawbacks are the non-automatic process

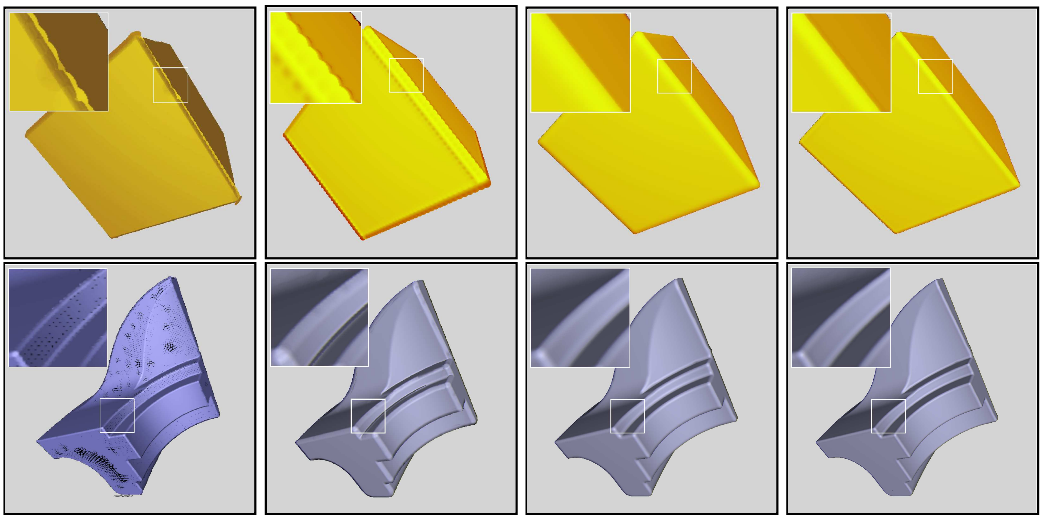


Figure 4.8: Surface approximations for the Cube and Fan Discs datasets. From left to right: EWA splatting, Adamson and Alexa's implicit surface definition, moving least-squares polynomial surface approximation (degree 2) and the implicit function based on approximate approximation (order $O(h^6)$).

to define the domain of the enriched functions and the need for investigating the properties of the enriched functions to ensure that the system of equations is invertible. Oriented normal vectors are also required by this method and, more importantly, a user-driven tagging of the points near sharp features.

Lipman *et al.* [104] presented a very interesting method that effectively represents singularities of the model. The method computes a *singularity indicator field* which intuitively assigns to each point an estimate of its proximity to a singularity. The computation of this field is based on the construction of a lower bound of the derivative at each sample point which is in turn based on the error expresion of the moving least-squares approximation previously presented by the authors [103]. Once the singularity indicator field is calculated, the singularity is approximated as an one-manifold by means of moving least-squares using the singularity indicator field to influence the weighting functions. This one-manifold is then used to generate either a discontinuous approximation or a continuous piecewise-smooth approximation. The discontinuous approximation is obtained by separating the space in two halves (limited by the one-manifold) and approximating the surface on both sides independently. The continuous piecewise approximation is generated by aligning one axis of the local coordinate system with the tangent line to the one-manifold at the origin of the local system and using the space of continuous piecewise bivariate polynomials of certain degree d for computing the approximation to the surface instead of the complete space of bivariate polynomials of degree d.

In this section, an efficient iterative method to represent sharp features is presented, which does not require any user intervention, based on bilateral filtering. This method is coupled with the proposed implicit surface definition based on approximate approximation. Bilateral filters have been previously used to robustly de-noise point sets by modifying either the sampled point positions (Mederos *et al.* [114]) or the normal vectors at the sampled points (Jones *et al.* [78]). The main difference between the method by Mederos *et al.* and the method by Jones *et al.* is that, while the former interprets the estimation of the normal as a minimization of a robust approximation formulation, the latter interprets it as the transformation of the normal vector given by the transposed inverse of the Jacobian matrix of the bilateral filter.

Similar results are used here to more accurately estimate normal vectors at points on the implicit surface. For this, consistently oriented normal vectors must be available at the sample points. If not, they can be precomputed. Traditionally, weighted covariance analysis has been used to estimate the normal vectors at the sample points when they are not available [5]. However, to represent sharp features using the approach presented here the input normal vectors at the sample points must be robust and noise-free. This can be achieved with the method by Mederos *et al.* [114] for denoising point clouds. In order to ensure a consistent orientation of the estimated normal vectors at the sample points, the method by Hoppe *et al.* [69] to orient normal vectors can be used.

4.3.1 Iterated AMLS implicits

The advantages of iterated approximate moving least-squares are twofold, firstly, it is possible to compute local approximations from scattered points without solving any linear system. Secondly, the number of iterations and the shape parameter can be set so as to produce an approximation that better fits the data but does not interpolate noise. Fasshauer and Zhang [48] present an iterative approach to estimate the optimal number of iterations and shape parameter. These advantages are exploited to define an implicit function $\mathcal{M}f_{AMLS} : \mathbb{R}^3 \to \mathbb{R}$, whose zero set approximates the input point set, as

$$\mathcal{M}f_{AMLS}(\mathbf{x}) = \sum_{i=1}^{N} g_i \varphi_i(\mathbf{x}) \tag{4.16}$$

where $g_i = \langle \mathbf{x}_i - \mathbf{x}, \mathbf{n}_i \rangle$ and $\mathbf{n}_i$ is the normal vector, given or estimated, at $\mathbf{x}_i$. This makes it possible to handle irregularly sampled point clouds and to provide a family of approximations ranging from approximate moving least-squares surfaces to radial basis functions interpolated surfaces. The iterative process for the

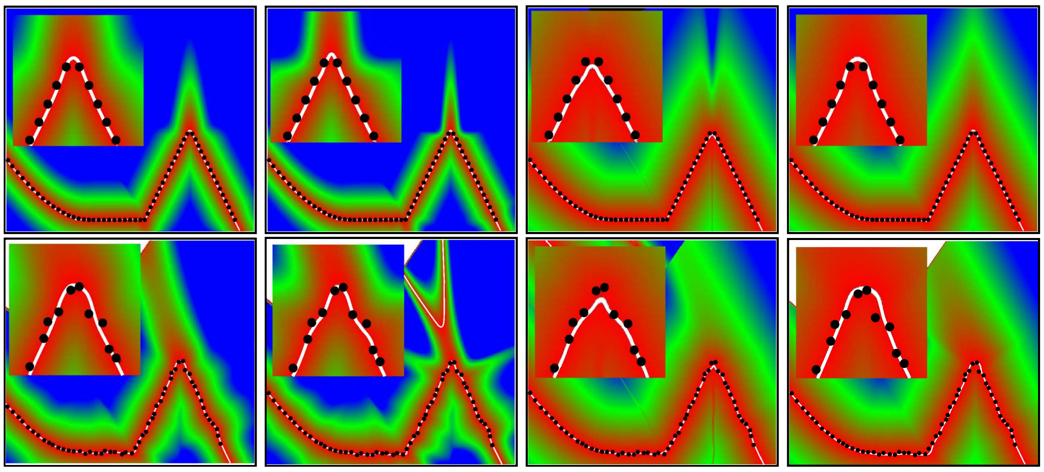


Figure 4.9: Plot of the value of the implicit function for a regularly (top) and an irregularly (bottom) sampled dataset. From left to right: AMLS implicits with 5 iterations, AMLS implicits with 20 iterations, Adamson and Alexa's implicits and Kolluri's implicits. The white line shows the zero set of the function while colors map the value of the implicit function with red corresponding to low values and blue to high values (see color plates).

surface approximation proposed is similarly defined as

$$\begin{aligned} \mathcal{M}f^{(0)}_{AMLS}(\mathbf{x}) &= \sum_{i=1}^{N} g_i \varphi_i(\mathbf{x}) \quad (4.17) \\ \mathcal{M}f^{(n+1)}_{AMLS}(\mathbf{x}) &= \mathcal{M}f^{(n)}_{AMLS}(\mathbf{x}) + \\ & \sum_{i=1}^{N} \left[g_i - \mathcal{M}f^{(n)}_{AMLS}(\mathbf{x}_i) \right] \varphi_i(\mathbf{x}). \quad (4.18) \end{aligned}$$

If g_i were not dependent of $\mathbf{x}$, it would be possible to pre-compute this iterated process in order to obtain the approximation by simply performing a weighted sum, as will be done in Chapter 6 for approximating volumetric data. Unfortunately this is not the case.

The direction of the normal vector at $\mathbf{x}$ can be also similarly stated as an iterative process as follows

$$\begin{aligned} \mathbf{n}^{(0)}_{AMLS}(\mathbf{x}) &= \sum_{i=1}^{N} \mathbf{n}_i \varphi_i(\mathbf{x}) \\ \mathbf{n}^{(n+1)}_{AMLS}(\mathbf{x}) &= \mathbf{n}^{(n)}_{AMLS}(\mathbf{x}) + \sum_{i=1}^{N} \left[\mathbf{n}_i - \mathbf{n}^{(n)}_{AMLS}(\mathbf{x}_i) \right] \varphi_i(\mathbf{x}). \end{aligned}$$

This is done, since convergence was not proved for the derivative of the iterated

	Adamson's	Kolluri's	AMLS implicits		
Model (points)	implicits	implicits	1 iter.	3 iter.	10 iter.
Stanford Bunny (35K)	30	29	103	107	126
Horse (48K)	50	52	183	283	389
Fan Disk (103K)	30	30	79	97	109
Armadillo Man (172K)	30	32	106	152	236
Stanford Dragon (400K)	116	120	361	499	778

Table 4.1: Performance measurements in seconds per frame for Adamson and Alexa's implicit formulation, Kolluri's implicit formulation and AMLS implicits. The performance increase in the Fan Disc and Armadillo Man datasets is due to the number of rays that effectively intersect the surface.

approximate approximation. That is, no theoretical guarantees are available that supports deriving Equations 4.17 and 4.18 to obtain a good reconstruction of the gradient. It is important to note that since g_i is fixed for each $\mathbf{x}$ during the whole iterative process, the fact that it depends on $\mathbf{x}$ does not affect the convergence of the method.

In the tests performed, for

$$r_i(\mathbf{x}) = \frac{\epsilon^3 \|\mathbf{x} - \mathbf{x}_i\|^2}{h^2},$$

Laguerre-Gaussian functions of order $O(h^2)$,

$$\varphi_i(\mathbf{x}) = \frac{\epsilon^3}{\pi^{3/2}} \exp\left(-r_i(\mathbf{x})\right), \tag{4.19}$$

order $O(h^4)$

$$\varphi_i(\mathbf{x}) = \frac{1}{\pi^{3/2}} \left(\frac{5}{2} - r_i(\mathbf{x})\right) \exp\left(-r_i(\mathbf{x})\right) \tag{4.20}$$

and order $O(h^6)$

$$\varphi_i(\mathbf{x}) = \frac{1}{\pi^{3/2}} \left(\frac{35}{8} - \frac{7}{2} r_i(\mathbf{x}) + \frac{1}{2} r_i(\mathbf{x})^2\right) \exp\left(-r_i(\mathbf{x})\right) \tag{4.21}$$

were used. Higher-order functions can be used, but the improvement would not be visually perceived and controlling their shape parameter is more difficult. One can easily generate the generalized Laguerre polynomials using mathematical software such as MATHEMATICA [167].

Comparisons with the implicit surface definitions by Kolluri [83] and by Adamson and Alexa [2] were performed for a two-dimensional synthetic point set (see

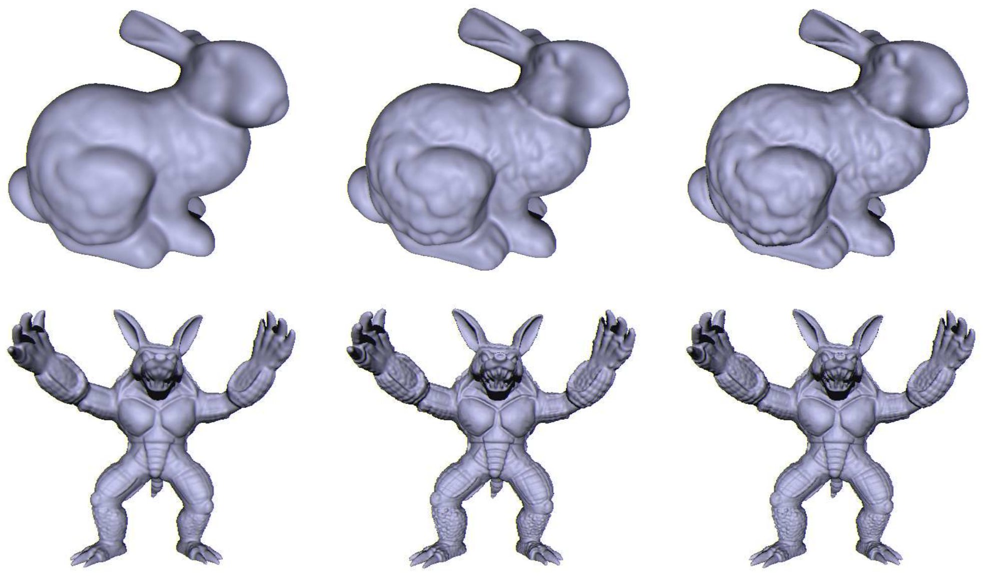


Figure 4.10: Renderings of the surfaces obtained with Adamson and Alexa's implicit function (left), Kolluri's implicit function (center) and AMLS implicits (right)

.

Figure 4.9). Irregularity was introduced into the point set to illustrate the adaptiveness of each method to these cases. Although Adamson and Alexa's and Kolluri's definitions are targeted to regularly sampled point clouds, the method based on AMLS is compared with them due to the fact that the simplicity of their definition match that of the AMLS implicits. Also, this helps to appreciate the impact on the performance caused by the iterative AMLS when compared to other similar implicit surface definitions.

As can be seen in Figure 4.9, the AMLS implicit function generates an approximation that fits the data more tightly without being truly an interpolant. The smoothness of the approximation can be controlled with the shape parameter and the number of iterations. In the detail windows of the figures, it can be appreciated that with 5 iterations, the approximation moves towards an interpolation but effectively filters noise. After 20 iterations however, the function fits the data even better and consequently starts to interpolate noise.

To perform the tests with three-dimensional models, a ray-tracing engine based on the methods by Wald and Seidel [162] and by Adamson and Alexa [3] was implemented. The ray-tracing implementation was tested with different point sets on a standard PC with a 3.4 GHz processor and 2 GB of RAM. Performance measurements were carried out with a 400×300 target viewport. Comparisons with the implicit surface definitions by Adamson and Alexa and by Kolluri were also

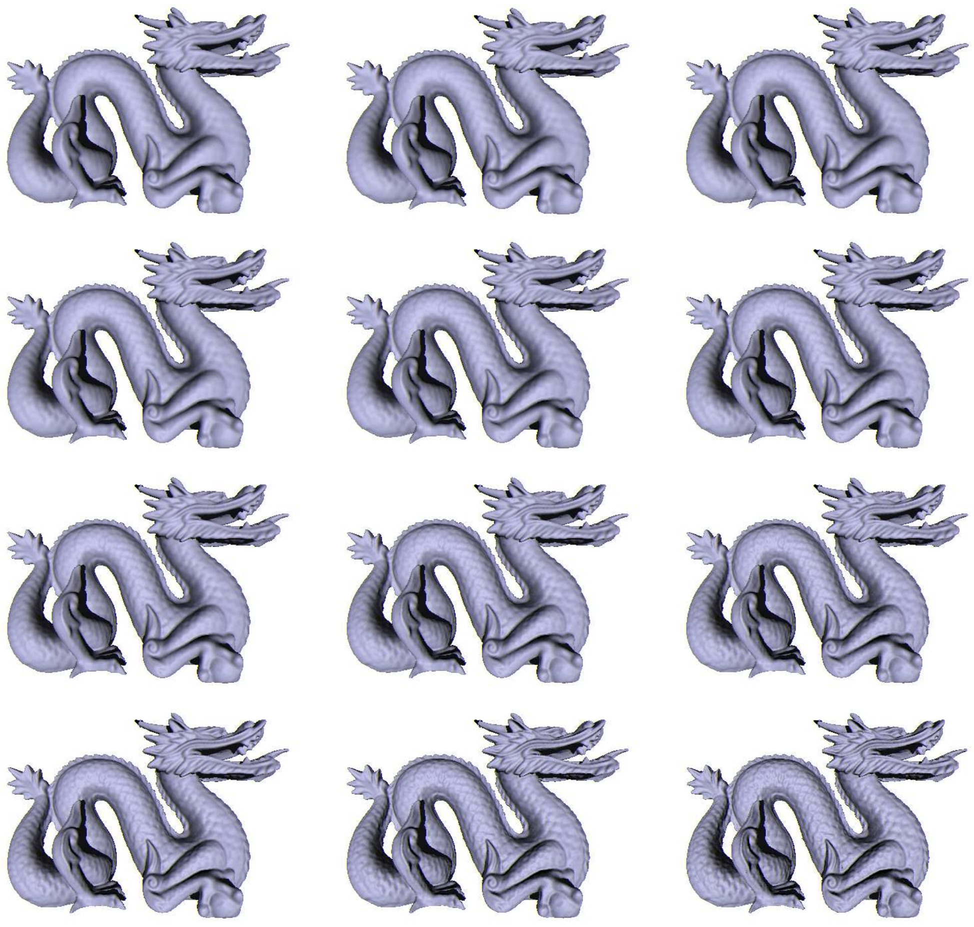


Figure 4.11: Effect of the number of iterations and the shape parameter ϵ on the approximation. From left to right: 3, 4 and 5 iterations. From top to bottom: $\epsilon = 0.4$, $\epsilon = 0.8$, $\epsilon = 1.22$, $\epsilon = 2.0$

performed for these models. In all cases, the same implementation of the raytracer was used, changing only the corresponding surface approximation method. The performance results in seconds per frame are shown in Table 4.1. As expected, the method presented here performs worse in terms of computational time compared to Adamson and Alexa's and to Kolluri's methods. This is due to the iterative process performed to correct the approximation so as to fit irregularly sampled data. Recall that the implicit definitions by Adamson and Alexa and by Kolluri are targeted at regularly spaced data. Also, the implicit function based on AMLS combines the simplicity of implicit formulations with the reconstruction quality of moving least-squares polynomial surface approximation and RBF implicit surface interpolation. A visual comparison among the three methods can be

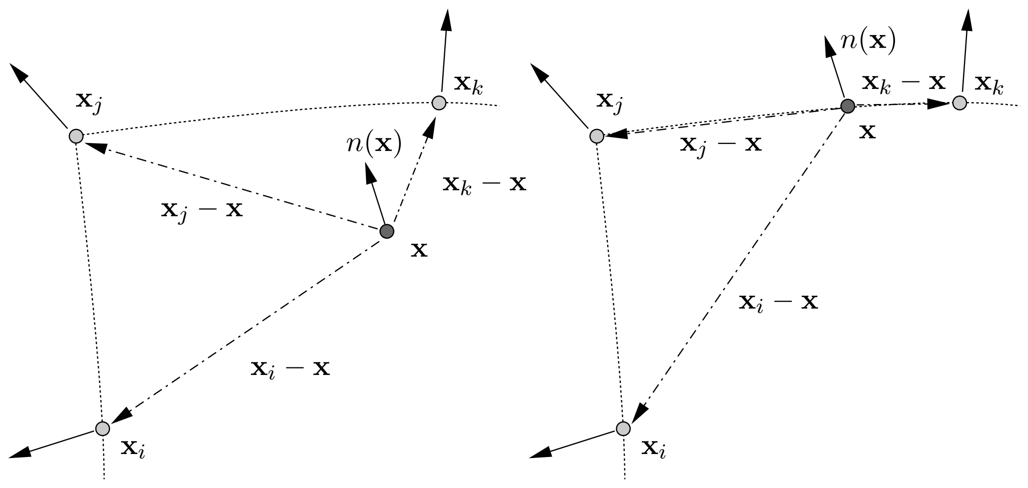

$\mathbf{x}_j$
$\mathbf{x}_k$
$n(\mathbf{x})$
$\mathbf{x}_k - \mathbf{x}$
$\mathbf{x}_j - \mathbf{x}$
$\mathbf{x}$
$\mathbf{x}_i - \mathbf{x}$
$\mathbf{x}_i$
$n(\mathbf{x})$
$\mathbf{x}_j$
$\mathbf{x}_k - \mathbf{x}$
$\mathbf{x}_k$
$\mathbf{x}_j - \mathbf{x}$
$\mathbf{x}$
$\mathbf{x}_i - \mathbf{x}$
$\mathbf{x}_i$

Figure 4.12: During normal vector estimation, weighing the points based on $\langle \mathbf{n}(\mathbf{x}), \mathbf{x}_i - \mathbf{x} \rangle$ produces good results only for points on or near the surface. At points not on the surface ($\mathbf{x}$ on the top figure) this measure may assign large weights to neighbors (*e.g.* $\mathbf{x}_i$) not expected to greatly influence the computations. For points on the surface ($\mathbf{x}$ in the right figure), on the other hand, the measure assigns larger weight to neighbors on the same plane as $\mathbf{x}$ or near to it ($\mathbf{x}_j$ and $\mathbf{x}_k$).

seen in Figure 4.10. It is important to mention that the same smoothing factor h was used for all methods. For the AMLS implicit function of the Stanford Bunny and of the Armadillo Man, 3 iterations with $\epsilon = 0.8$ were performed. It can be seen that the AMLS implicit surface better approximates the sample points. However, for low-sampled models, such as the Stanford Bunny, some regions with flat spots appear when too many iterations are performed or the shape parameter is set too high. Another interesting comparison is shown in Figure 4.8, where it can be seen that, by using higher-order generating functions, the resulting approximation fits the surface more tightly. In order to show the effect of the shape parameter and the number of iterations on the approximation, tests with different values for ϵ were carried out and the surface was rendered at each iteration. The resulting renderings are shown in Figure 4.11. As can be seen, ϵ affects the convergence ratio of the iterative process. Therefore, both parameters must be estimated at the same time as described by Fasshauer and Zahn [48].

4.3.2 Introducing sharp edges

Here, a method that automatically approximates sharp edges is provided, which is easy to understand and to implement. This method is based on the robust estimation of normal vectors. The approach proposed to robustly estimate the normal vectors, at any point on the implicit surface, is based on the minimization of the

Model (points)	Adamson's implicits	MLS (degree 2)	Approx. $O(h^6)$	Approx. sharp edges
Cube (6K)	6	140	14	24
Fan Disc (26K)	17	234	21	32
Stanford Bunny (35K)	20	503	30	45
Armadillo Man (173K)	16	122	17	23

Table 4.2: Performance measurements in seconds per frame for Adamson and Alexa's implicit formulation, moving least-squares surfaces with local polynomial approximations and the method proposed without and with bilateral filtering for approximating sharp edges. The performance increase in the Armadillo dataset is due to the number of rays that effectively intersect the surface. Note that, differently from the results presented above, the iterated method was not performed here, which results in lower computational times. Furthermore, a smaller viewport was used, which accounts for the difference in the results for Adamson's method.

following expression

$$\sum_{i=1}^{N} \vartheta(g_i)\omega_{MLS}(\mathbf{x}, \mathbf{x}_i) \tag{4.22}$$

with respect to g_i, where $g_i = \langle \mathbf{n}, \mathbf{x}_i - \mathbf{x} \rangle^2$,

$$\vartheta(t) = 1 - \exp\left(-\frac{t}{2\sigma_\vartheta}\right)$$

and σ_ϑ is a parameter that controls the sensitivity of the expression to outliers and edges, subject to the restriction $||\mathbf{n}|| = 1$. The robustness is introduced by the function ϑ. Other functions ϑ, as well as their properties, are given in the work by van de Weijer and van den Boomgaard [155]. The normal vector $\mathbf{n}$ obtained with this minimization provides an estimative that approximates sharp edges, since the sample points near the plane defined by the normal vector $\mathbf{n}$ and the point $\mathbf{x}$ will have a greater influence in the estimative (see Figure 4.12).

To minimize this equation, its derivative is used, which results in a non-linear system, which can be computationally expensive to solve. Therefore, the fact that the normal vectors at the sample points were already estimated (see previous section) is exploited to define an iterative process to approximate the normal vectors at points on the surface. Given a first estimative $\mathbf{n}^{(0)}(\mathbf{x})$ of the normal vector at $\mathbf{x}$, the normal vector approximation can be refined using the following iterative process:

$$\mathbf{n}^{(n+1)}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \mathbf{n}_i \varphi_i^{(n)}(\mathbf{x})}{\| \sum_{i=1}^{N} \mathbf{n}_i \varphi_i^{(n)}(\mathbf{x}) \|}. \tag{4.23}$$

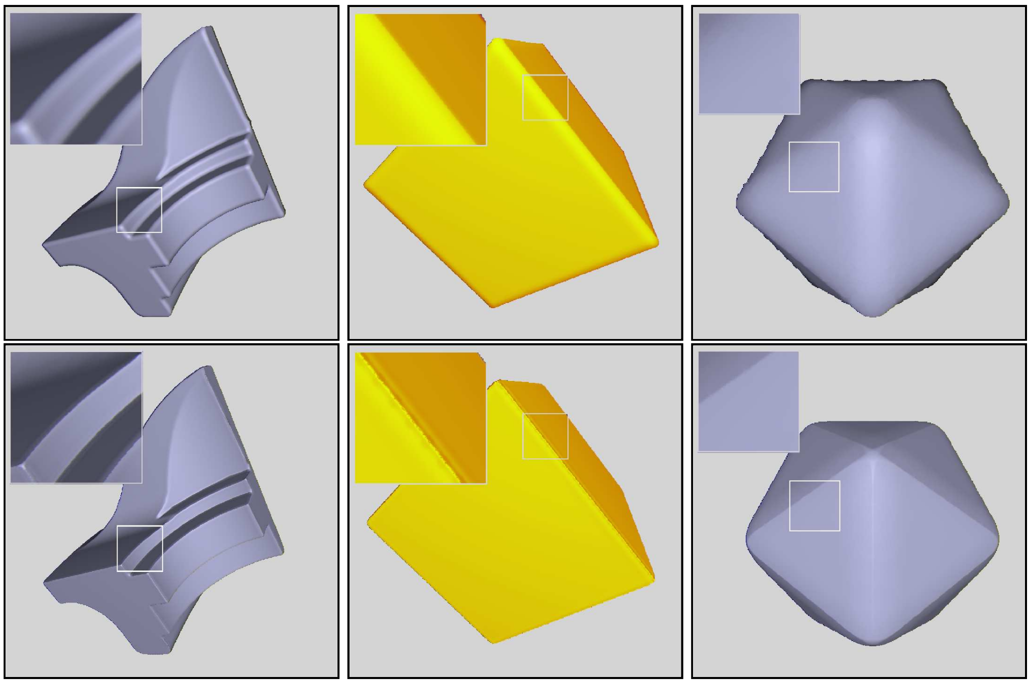


Figure 4.13: Renderings of the Fan Disc, Cube and Icosahedron datasets with moving least-squares polynomial surface approximation (top) and the method proposed with sharp edges enhancement (bottom).

Here, $\varphi_i^{(n)}$ is defined as follows. Since the derivative of ϑ, obtained from deriving Equation 4.22 to minimize it, results in a robust weight

$$\omega_{RMLS}^{(n)}(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{\langle \mathbf{n}^{(n)}(\mathbf{x}), \mathbf{x}_i - \mathbf{x}\rangle^2}{2\sigma_\vartheta}\right),$$

the radial weight used for the generating function is given by

$$\omega_{MLS}(\mathbf{x}, \mathbf{x}_i)\omega_{RMLS}^{(n)}(\mathbf{x}, \mathbf{x}_i),$$

which leads to

$$\varphi_i^{(n)}(\mathbf{x}) = \frac{\exp\left(-s_i^{(n)}(\mathbf{x})\right)}{\pi^{3/2}}\left(\frac{35}{8} - \frac{7}{2}s_i^{(n)}(\mathbf{x}) + \frac{1}{2}s_i^{(n)}(\mathbf{x})^2\right),$$

where

$$s_i^{(n)} = \frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{h^2} + \frac{\langle \mathbf{n}^{(n)}(\mathbf{x}), \mathbf{x}_i - \mathbf{x}\rangle^2}{2\sigma_\vartheta}.$$

This is done in the spirit of *bilateral filters* [44], and of the results of the work by Fenn and Steidl [49], who derived an iterative process for robust data approximation based on the work by van den Weijer and van den Boomgaard [155].

Note that, although each iteration of this process is an approximate moving least-squares approximation, the robust formulation of Equation 4.22 is not a moving least-squares approximation. In the experiments performed, 5 iterations sufficed in all cases to obtain a good approximation to the sharp feature.

One important issue of the iterative normal vector improvement process given by Equation 4.23 is that the point $\mathbf{x}$ where the normal vector is estimated must be on or near the surface as shown in Figure 4.12.

To perform the tests, this process was included in the ray-tracing engine. The performance results in seconds per frame are shown in Table 4.2. As before, compared to the implicit definition by Adamson and Alexa, the method presented here performs slightly worse. However, when the approach presented here based on bilateral filtering is used, the method is able to approximate sharp edges with low additional computational effort. The results are shown in Figure 4.13, where the method proposed for sharp edges approximation is compared with moving least-squares polynomial surface approximation. As can be seen, the bilateral filter corrects the normal vectors near the edges while maintaining smooth areas unaltered. Even for the low-sampled Icosahedron model with only 600 points and wide angles, the method proposed was able to approximate sharp edges. However, note that, although sharp edges are preserved, corners are slightly smoothed in this same model. This might be due to the fact that the points in the neighborhood of the corner lay on three (or more) intersecting planes, which reduces the effect of the bilateral filters. Note that this normal vector correction process is applied only on points on the surface. Thus, since the actual surface definition is based on a smooth vector field, the C^0-continuity of the approximate surface is mantained.

4.4 Adaptive Partition of Unity Implicits

Among the surface reconstruction methods based on implicit functions, methods based on partition of unity have been recently used due to their nice properties concerning processing time, reconstruction quality and capacity to deal with massive data sets [122; 113]. Basically, these methods determine a domain subdivision for which local functions are computed and combined to define a continuous global implicit approximation for the point set.

In this section, a method [58] developed to tackle important issues from previous surface reconstruction techniques based on partition of unity is described. The proposed surface reconstruction method effectively combines the space subdivision with an adaptive construction of local polynomial approximations by means of multivariate orthogonal polynomials [18]. This makes it possible to increase the degree of the polynomial at locations where higher-degree polynomials are needed to obtain a good approximation to the surface. Contrary to previous methods, the iso-surface is extracted directly from the data structure used to subdivide

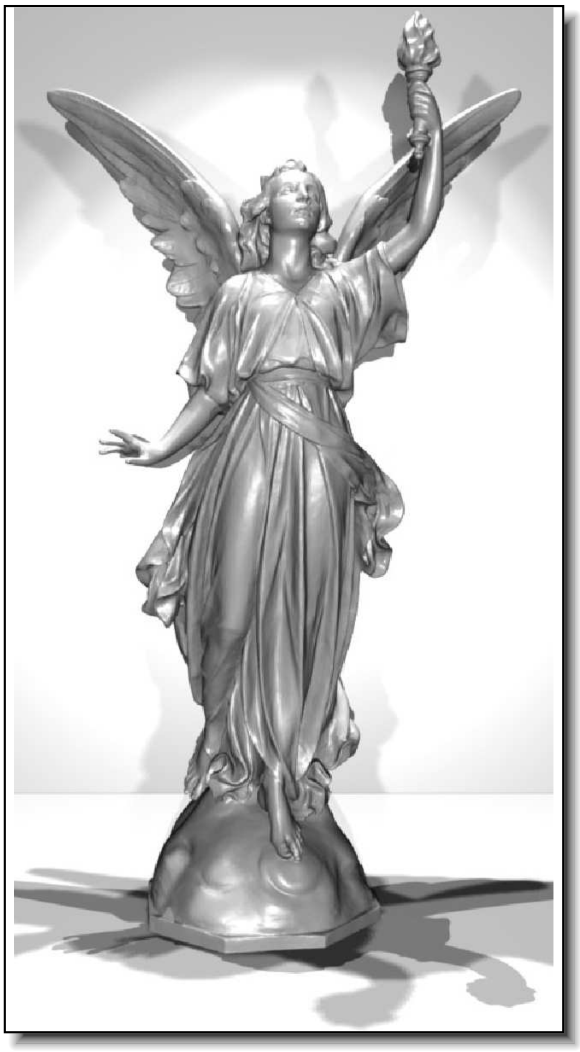


Figure 4.14: Stanford Lucy (16M Points) reconstructed with the proposed method.

the space, namely, the J^1_A . This enables the adaptive surface extraction to take advantage of refinement information obtained during function approximation. Furthermore, as the J^1_A is composed of tetrahedra, the surface extraction algorithm guarantees topologically coherent surfaces. The aspect ratio of the triangles generated by means of J^1_A polygonization is usually poor, which motivates the use of a simple, but effective, J^1_A vertex displacement technique that is able to considerably improve mesh quality.

In general, least-squares formulations solved by normal equations using canonical polynomials lead to ill-conditioned systems of equations. By using a basis of orthogonal polynomials, no system of equations must be solved and stability is improved without requiring expensive computations in contrast to other methods such as QR decomposition with Householder factorization, singular value decomposition or pre-conditioned conjugate gradient. Furthermore, the use of orthogonal polynomials makes it possible to efficiently increase the degree of the local polynomial approximation due to the recursive nature of their construction. For this reason, the method proposed is also adaptive with respect to the degree of the local fittings.

Contrary to previous work on multi-level partition of unity implicits, the proposed method is able to avoid generating spurious surface sheets and surface artifacts. This is achieved by using tests that discard approximations considered non-robust, *i.e.*, approximations which oscillate within the local support. However, in some cases, the automatic reconstruction algorithm may cause loss of details in some regions due to the use of low-order polynomial functions. Moreover, even with good robustness criteria, partition of unity implicits are particularly sensitive to noise. As before, changing the robustness conditions to tighten the solution may lead to over-smoothing effects without assuring the removal of all problems. Thus, allowing the user to locally edit the global approximation to place more suitable shape functions is an interesting feature proposed here.

This machinery allows the definition of a two-fold adaptive partition of unity method in the sense that both the degree of the local approximation and the space subdivision are adapted to better fit the surface. An example of a surface reconstructed with the proposed method is depicted in Figure 4.14.

4.4.1 Multi-level partition of unity implicits

Partition of unity implicits are defined on a finite domain Ω as a global approximation $\mathcal{M}f_{PU}$ obtained with a linear sum of local approximations. As with other implicit surface approximation methods, the surface is defined as the zero set of $\mathcal{M}f_{PU}$. For this purpose, a set of non-negative weight functions $\Phi = \{\phi_1, \dots, \phi_K\}$ with compact support, where $\sum_{i=1}^{K} \phi_i(\mathbf{x}) \equiv 1$, $\mathbf{x} \in \Omega$, and a set $\mathcal{F} = \{f_1, \dots, f_K\}$ of local signed distance functions f_i must be defined on Ω. Given the set $\mathcal{F}$ and Φ, the function $\mathcal{M}f_{PU} : \mathbb{R}^3 \to \mathbb{R}$ is defined as:

$$\mathcal{M}f_{PU}(\mathbf{x}) \equiv \sum_{i=1}^{K} f_i(\mathbf{x})\phi_i(\mathbf{x}), \quad \mathbf{x} \in \Omega. \tag{4.24}$$

A set of non-negative functions $\Theta = \{\theta_1, \dots, \theta_K\}$ with compact support produces the partition of unity as

$$\phi_i(\mathbf{x}) = \frac{\theta_i(\mathbf{x})}{\sum_{k=1}^{K} \theta_k(\mathbf{x})},$$

where θ_i is a weight function with compact support. The domain Ω is covered by a set of supports and, for each one, a function f_i and a weight function ϕ_i

are defined. Otahke *et al.* subdivide the domain using an octree and define a spherical support for each cube. The functions $f_i : \mathbb{R}^3 \to \mathbb{R}$ at each local support are computed using the set of points in the support by initially defining a local coordinate system (ξ, η, ν) at the center of the support where (ξ, η) define the local plane (domain) and ν coincides with the orthogonal direction (image). Hence, f_i is defined as $f_i(\mathbf{x}) = w - g_i(u, v)$, where (u, v, w) is $\mathbf{x}$ in the (ξ, η, ν) basis. The function g_i is obtained by the two-dimensional least-squares method. Note that this method requires points equipped with consistently oriented normal vectors.

4.4.2 The J^1_A triangulation

Castelo *et al.* [29] proposed the J^1_A triangulation as an adaptive structure with an underlying algebraic description that allows both efficient memory usage and the ability of being defined in any dimension. Such algebraic description is based on two mechanisms: the first is used to uniquely identify a simplex within the triangulation, and the second is used to allow traversals in the structure.

The J^1_A triangulation is conditioned by a grid of blocks which correspond to n-dimensional hypercubes in $\mathbb{R}^n$. The J^1_A simplices are obtained by the division of such blocks in a way that each simplex is coded by the 6-tuple $S = (g, r, \pi, s, t, h)$. The first two components are related to the location of the block within the grid, whereas the last four identify the simplex within the block. Specifically, the n-dimensional vector g provides the location of a particular block in a particular refinement level r. Figure 4.15 illustrates, on the left, a two-dimensional J^1_A and, on the right, a highlighted block of refinement level $r = 0$ (0-block) and $g = (3, 1)$. Also in Figure 4.15, one can find blocks with refinement level $r = 1$ (1-block) depicted in dark blue.

Before explaining how simplices are described, it is important to mention that an n-dimensional J^1_A allows the refinement of an i-block by splitting it into 2^n $(i + 1)$-blocks. It is also worth to notice that, in order to accommodate the newly created blocks, some other blocks may be forced to be refined so that the difference in the refinement level of two neighboring blocks never becomes greater than one. The last part of this accommodation process is to impose that whenever blocks whose refinement levels differ by one are neighbors, the one having the smallest r is transformed into a transition block. Such a block is the one that possesses only some of its k-dimensional faces $(0 < k < n)$ refined. The situation is illustrated by Figure 4.15 in which basic 0-blocks are colored light-blue, basic 1-blocks are colored dark-blue and transition blocks are colored light-red. In particular, the highlighted transition block has only its left edge refined. From now on, for the sake of clarity, non-transition blocks will be referred to as basic blocks.

The simplex representation is based upon the fact that all simplices in a block share at least the vertex v_0, which is the center of the n-dimensional hypercube. So, starting in v_0, the next step is taken in the positive or negative direction of

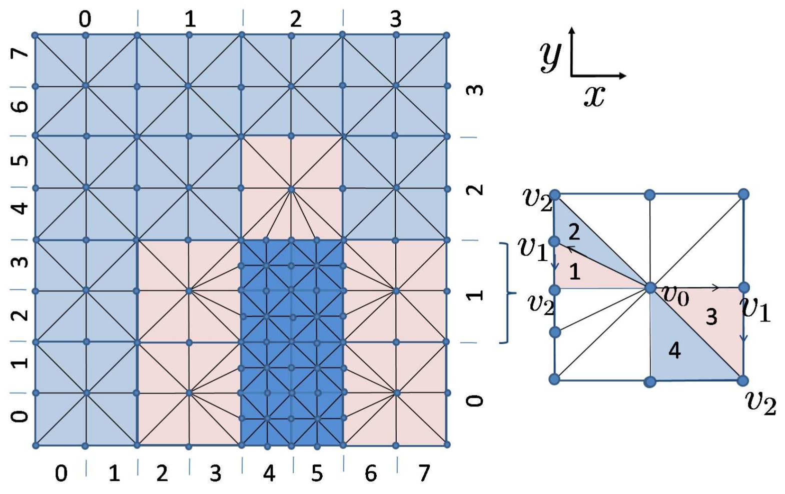


Figure 4.15: The J_A^1 triangulation: on the left, a sample two-dimensional adaptive triangulation and, on the right, examples of pivoting operations (see color plates).

one chosen coordinate axis. This will produce v_1 as the center of an $(n-1)$-dimensional face and, as the process continues, the vertices $v_2 \ldots v_n$ will be defined as the center of $(n-2), \ldots, 0$ dimensional faces respectively. In other words, simplices can be represented by the path traversed from v_0 to v_n which is coded by π and s. The π vector stores a permutation of n integers from 1 to n representing coordinate axes, while s represents the direction, positive or negative, that must be followed in each axis. For instance, in Figure 4.15, Simplex 3 is represented by $\pi = (1, 2)$ and $s = (1, -1)$, which means that, from v_0, the path is traced through axis $\pi_1 = 1$ (x, in the figure) in the positive direction ($s_{\pi_1} = 1$) and then through axis $\pi_2 = 2$ (y, in the figure) in the negative direction ($s_{\pi_2} = -1$).

For simplices inside basic blocks and simplices inside transition blocks that do not reach any refined face, the information provided by π and s suffices. However, in the remaining cases, further information must be provided, because when a refined k-dimensional face is reached, there is not only one center, but 2^k centers. For this reason, the scalar h is used to inform how many steps are taken before a refined face is reached, while vector t defines extra signs for axis $\pi_{h+1} \ldots \pi_n$ that are used for selecting one center from all possibilities. For instance, in Figure 4.15, Simplex 1 is represented by $\pi = (1, 2)$, $s = (-1, -1)$, $h = 1$ and $t = (0, 1)$ because only one step is taken before reaching a refined edge and the chosen center for placing v_1 is in the positive direction of π_{h+1}.

Besides describing the location of simplices, the J_A^1 triangulation also defines pivoting rules for traversing the triangulation without using any other topological data structure. These rules are completely algebraic in that they take a simplex description (S', for instance) as input and outputs another simplex description

(S'') as the result of the pivoting operation. Figure 4.15 illustrates two pivoting operations in which simplices 1 and 2 are pivoted in relation to vertices v_2 and v_1 respectively, generating simplices 3 and 4. All pivoting rules can be found in the work by Castelo *et al.*

4.4.3 Robust adaptive partition of unity implicits

Traditionally, adaptive partition of unity implicits are built using an octree to partition the space and calculate local approximations that are subsequently combined using weights. The more details the object possesses, the more refined the octree must be. Thus, the octree can be used to identify complicated or soft features on the surface. Hence, the goal was to use this information acquired during function approximation to obtain an adaptive polygonization. Therefore, as J_A^1 has an underlying restricted octree as its backbone, the triangulation was adapted to serve both approximation and polygonization purposes. The achieved adaptiveness allows capturing fine details without using refined grids. Another important feature of the method is the increased quality of the local approximations compared to previous methods, which prevents spurious sheets and surface artifacts.

Local approximations. The local approximations $f_i : \mathbb{R}^3 \to \mathbb{R}$ are generated in the spherical supports of the weight functions ϕ_i, which are defined as the circumspheres of the blocks enlarged by a factor greater than one (*e.g.,* 1.5).

The function f_i at each block is computed using the set of points encompassed by its support by initially defining a local coordinate system (ξ, η, ν), as explained before, at the center of the support. Recall that f_i is defined as $f_i(\mathbf{x}) = w - g_i(u, v)$, where (u, v, w) is $\mathbf{x}$ in the (ξ, η, ν) basis.

In the method proposed, the local functions are approximated by means of polynomial least-squares fitting. However, instead of using a canonical basis $\{u^i v^j : i, j \in \mathbb{N}\}$, a basis of orthogonal polynomials with respect to the inner product induced by the normal equation is chosen. This way, it is not necessary to solve any system of equations. To find such a basis, the method by Bartels and Jezioranski [18] is used.

Then, given a set of orthogonal polynomials $\Psi = \{\psi_1, \ldots, \psi_M\}$, the polynomial fitting in local coordinates can be computed as:

$$g_i(u,v) = \sum_{j=1}^{M} \psi_j(u,v) \frac{\sum_{\mathcal{S}_i} w_k \psi_j(u_k, v_k)}{\sum_{\mathcal{S}_i} \psi_j(u_l, v_l)\psi_j(u_l, v_l)}; \; i = 1, \cdots, K, \tag{4.25}$$

where $\mathcal{S}_i \equiv \operatorname{supp}(\mathbf{c}_i, R_i)$, $\mathbf{c}_i$ and R_i are the center and radius of the block i respectively, and (u_i, v_i, w_i) is $\mathbf{x}_i$ in the basis (ξ, η, ν). Thus, g_i provides an approximation to the solution of

$$\min_g \sum_{(u_i, v_i)} (g(u_i, v_i) - w_i)^2.$$

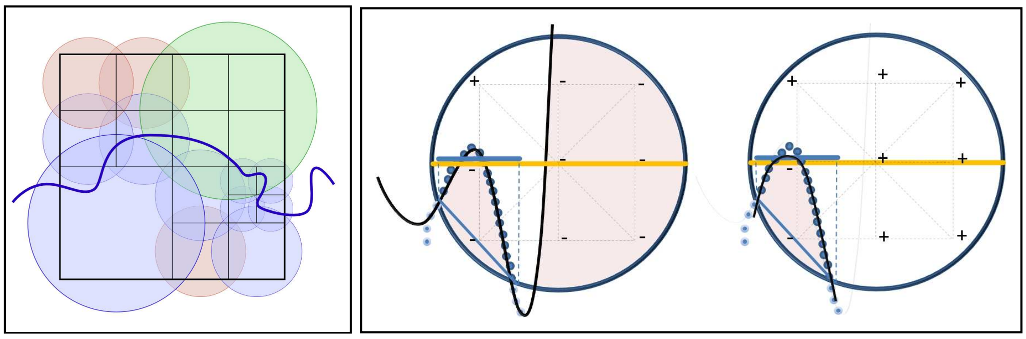


Figure 4.16: The figure in the left side depicts the behavior of the J_A^1 during function approximation. The figure in the right side shows an illustration of the effect of the coverage domain on the polynomial approximation. The left side of this figure depicts a case that can arise when a high-order polynomial is used to approximate the surface inside the local domain (blue circle). Since a large region of this domain is void of points, the polynomial approximation may oscillate. Thus, the coverage domain (blue line) is computed and the ratio of the area of the coverage domain and the area of the plane (yellow line) is calculated. This ratio determines the degree of the polynomial used. Decreasing the degree of the polynomial when this ratio is below a threshold reduces the oscillation as can be seen on the right side of the figure (see color plates).

The main motivation for using such orthogonal polynomials is their ability of generating higher-degree approximations from previously computed approximations with low additional computational effort. As stated before, this property enables the definition of a method that is adaptive not only in the spatial subdivision but also in the local approximation. However, a downside of employing high-degree polynomials as local solutions is the fact that such functions may present oscillatory behavior and, even if they present a small least-squares error, they may be a poor approximation inside the region of interest. For instance, in Figure 4.16 on the left, the polynomial is close to the sample points inside the support, but the signs obtained when the function is evaluated at the vertices of the J_A^1 block are not correct. Therefore, depending on the neighbor solutions, this situation may lead to extra surface sheets or artifacts.

Before presenting a solution, one must notice that this problem occurs because even though high degree polynomials are able to approximate point data nicely in-between points, they can also oscillate at locations in which there are not points to restrain the solution, as illustrated on left side of Figure 4.16. Based on this fact, it can be observed that the distribution of points inside the support is as important for generating a good function as the number of points used in the least-squares minimization. Figure 4.18c and Figure 4.18d depict a real application of the coverage domain.

Thus, an approximate, but computationally inexpensive, way to assess how well the points are distributed inside the support is presented. As in the proposed method the local domains are actually planes, it is necessary to determine how large is the area of these planes covered with points. For this, the ratio (k) between the projection of the support of the block and the projection of the bounding box of the points over the plane is calculated. To that end, a coverage criterion for polynomials was created by establishing a minimum value of k for each polynomial degree (for instance, 0.4, 0.8 and 0.85 can be used as the default parameters for second, third and fourth degree polynomials). In Figure 4.16 the importance of the coverage domain for aiding the choice of the correct function is illustrated, since, in the example, a lower degree approximation (on the right) is more suitable than a higher one (on the left).

As mentioned before, besides the coverage criterion, it is also necessary to take into account the minimum number of points that should be used for each polynomial degree (the default is twice the number of polynomial bases for each degree). The union of both of these criteria constitute the robustness test used in the proposed method.

An immediate issue that arises from this minimum point test is that not every block in the domain encloses enough points for the polynomial approximation described previously. Therefore, we created a strategy for handling blocks with few or without points that differs from previous work because, instead of iteratively growing the support of the block until the minimal number of points is reached [122], which may cause a local approximation to influence a large part of the domain, or using the approximation of the parent of the block [113], which can be a poor approximation, this situation is addressed by searching the nearest cluster of points to the current block and performing a first degree approximation.

Such cluster of points is determined by finding the nearest sample point $\mathbf{r}$ to the center of the block, by querying enough neighbors around $\mathbf{r}$ (the default is 20 points), and by approximating a least-squares plane. However, depending on the point distribution, the plane can be orthogonal to the expected resulting plane [10]. This situation is detected by comparing the normal vector of the plane with the average of the normal vectors of $\mathbf{r}$ neighbors. If the angle is greater than $\pi/6$, the least-squares function is substituted by the plane with the normal vector equals to the computed average and the origin equals to the average neighbor position. This last test is considered as the third robustness condition of the method.

Now that the most important concepts of the proposed approach were clarified, an algorithmic outline of the method is provided: after setting up the initial J_A^1 configuration, for each block that does not have an approximation defined, the number of points inside the support of the block is queried, originating three different situations: (i), the number of points is enough for performing approximations; (ii), the number of points is not enough even for a least-squares plane; and (iii), the number of points is greater than a specified maximum threshold.

Initially, in case (i), a test that measures the variation of point normal vectors by Ohtake *et al.* [122] is used to determine the presence of two or more surface sheets inside the same support. If they exist, the method refines the block and the process starts again for the newly created blocks. If only one sheet is detected, a polynomial with degree one is calculated and its degree is recursively increased until the error criterion is met, or until the robustness test does not allow a higher degree or until the degree of the polynomial is equal to the maximum allowed (four as default). If the previous process finishes and the error is acceptable, the approximation is stored in the block, otherwise, the block is refined, unless its support possesses a critical number of points (*e.g.*, less than 100). In this situation, the subdivision may be aborted if the new approximation blocks would increase the error instead of decreasing it, due to the fact that the new blocks would enclose small amounts of points that would not allow high degree approximations.

In both parts of the above description in which the refinement is suggested, the block may be already in the user-defined maximum allowed refinement level, so there is no other option rather than using the best approximation calculated so far.

The approximation case (ii) is handled by searching the nearest cluster of points from the current block and performing a first degree approximation as explained above. Finally, case (iii) in a heuristic employed to avoid useless and expensive calculations. It is an unnecessary effort to calculate minimizations for more than one thousand points. Thus in this case, subdivision of the block is forced whenever the maximum refinement was not reached, otherwise the approximation is computed anyway.

This section is concluded by elucidating the difference between block refinements caused by approximation conditions and those triggered by J_A^1 restrictions (explained in Section 4.4.2). In the latter case, new approximations do not have to be computed if the approximation for the block being refined is already good. Figure 4.16 illustrates a case in which not all leaf nodes hold approximations associated to them. In the figure, the blue circles represent leaf blocks that hold approximations and points inside them, the orange ones are also leaves that hold approximations despite of not having points in their supports, and the green one stands for a non-leaf node that holds the approximation and was only divided due to the J_A^1 accommodation process.

Function evaluation and adaptive polygonization. Given a point $\mathbf{x}$ inside the domain, an octree-like traversal of the J_A^1 blocks is conducted to determine which blocks encompass $\mathbf{x}$ within their supports. The value of $\mathcal{M}f_{PU}(\mathbf{x})$ is obtained as a combination of all local functions from the supports found to contain $\mathbf{x}$:

$$\mathcal{M}f_{PU}(\mathbf{x}) = \frac{\sum_{i=1,\mathbf{x}\in\mathcal{S}_i}^{M} f_i(\mathbf{x})\theta_i(\mathbf{x})}{\sum_{i=1,\mathbf{x}\in\mathcal{S}_i}^{M} \theta_i(\mathbf{x})},$$

where $\theta_i(\mathbf{x}) = \theta(\|\mathbf{x} - \mathbf{c}_i\|/R_i)$ and the weight function $\theta(t) = (1 - t^2)^4$ that assumes zero value for $t \geq 1$.

The polygonization finds first an initial simplex, which is a straightforward task when a point near the surface is available. Then, a traversal of all simplices that intersect the surface through J_A^1 pivoting rules is conducted, while generating the adaptive surface mesh.

4.4.4 Extensions to the method

Here, two extensions to the method above presented are proposed. The difficulty in handling parameters in most surface reconstruction approaches suggests that in some cases it may be useful to allow manual edition of the function so that the user can either fix undesirable artifact or enhance the approximation quality by choosing more appropriated functions. This constitutes the first extension to the method.

The second extension is related to the low quality presented by the meshes generated by the J_A^1 polygonizer. It is actually an computationally inexpensive and memory efficient technique that displaces J_A^1 vertices in order to enhance the aspect ratio of the triangles.

Interactive implicit function editing. For the function editing feature, one of the advantages presented by partition of unity implicits methods over other techniques, such as moving least-squares or radial basis functions, is exploited, namely, the fact that the global function is defined by a set of independent local functions in subdomains, *i.e.*, changing one of these functions only affects the global function locally. Thus, changing local traits of the function consists in locating the desired block and switching the associated approximation. Finding such a block in the J_A^1 triangulation is quite straightforward, since its structure consists of a restricted octree. The calculation is made by testing the blocks that contain the desired point, in different refinement levels, until the one with the function is found.

In the implementation used, a graphic tool was developed for choosing points over a reconstructed model and selecting either a new degree for the polynomial approximation or a constant function, which can be defined by the user or automatically computed.

Mesh enhancement. To approach the problem of the poor quality of some triangles generated from the J_A^1 , the mesh displacement technique was chosen because it is quite simple to implement and does not impose a heavy overhead on the time and memory complexities of the polygonizer. However, as the degrees of freedom for moving J_A^1 vertices without invalidating the structure are constrained, the improvement is limited. The idea is to move the J_A^1 vertices away from the surface a distance inversely proportional to the function value, in order to improve the aspect ratio of the mesh elements.

The displacements are applied only on J_A^1 vertices belonging to simplices that

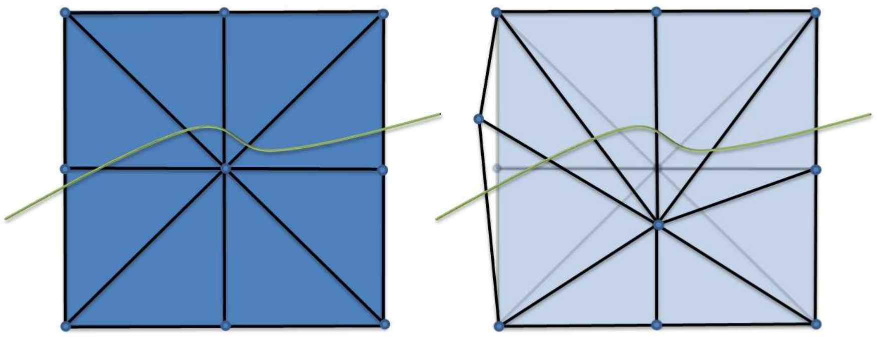


Figure 4.17: Illustration of the deformation of a J_A^1 block: the vertex displacement technique is able to create more uniform mesh elements.

intersect the surface during the polygonization. For that reason, no extra memory is needed and the extra computational effort is due to the calculation of an approximation for the maximum value of the function and for evaluating the gradient of the function at every displaced vertex. The following equation presents how a vertex $\mathbf{x}$ is taken to its new position $\mathbf{x}_{new}$

$$\mathbf{x}_{new} = \mathbf{x} + \left[\frac{\operatorname{sign}\left(\mathcal{M}f_{PU}(\mathbf{x})\right)}{\|\nabla \mathcal{M}f_{PU}\|} \left(\frac{\mathcal{M}f_{PU}^{\max} - \mathcal{M}f_{PU}(\mathbf{x})}{\mathcal{M}f_{PU}^{\max}} \right)^2 \frac{e(l)}{4} \alpha \right] \nabla \mathcal{M}f_{PU}(x)$$

where $\operatorname{sign}\left(\mathcal{M}f_{PU}(\mathbf{x})\right)$ represents the function sign at $\mathbf{x}$, $\mathcal{M}f_{PU}^{\max}$ is an estimative to the maximum of the function in the domain, l is the refinement level associated to the vertex, $e(l)$ is the size of a J_A^1 block edge in the refinement level l, and $0 < \alpha < 1$ determines the maximum amplitude of the movement.

For basic blocks, the refinement level associated with a vertex l is equal to the refinement of the block r, whereas, for transition blocks, it is equal to $(r + 1)$ for vertices that lie in refined faces and equal to r for the remaining ones. It is worth to mention that $e(l)/4$ is the maximum amplitude of the displacement, in any direction, allowed by the J_A^1 structure. In Figure 4.17 a representation of the displacement of the vertices is depicted in two dimensions.

In Figure 4.18, the importance of the coverage domain is illustrated, as well as the interactive function editing procedure. In Figure 4.18a, the 362K raw data Stanford Bunny was reconstructed using the method by Ohtake *et al.* using its default parameters, while in Figure 4.18b, we selected the parameters to match those used with the method proposed (maximum refinement 9 and error threshold 0.002). Note that this set of points is the raw Stanford Bunny dataset (362K points), whereas the Stanford Bunny dataset used in previous sections contains the vertices of the reconstructed versions (35K points). In Figure 4.18c, the proposed technique was used to reconstruct the same model without the coverage

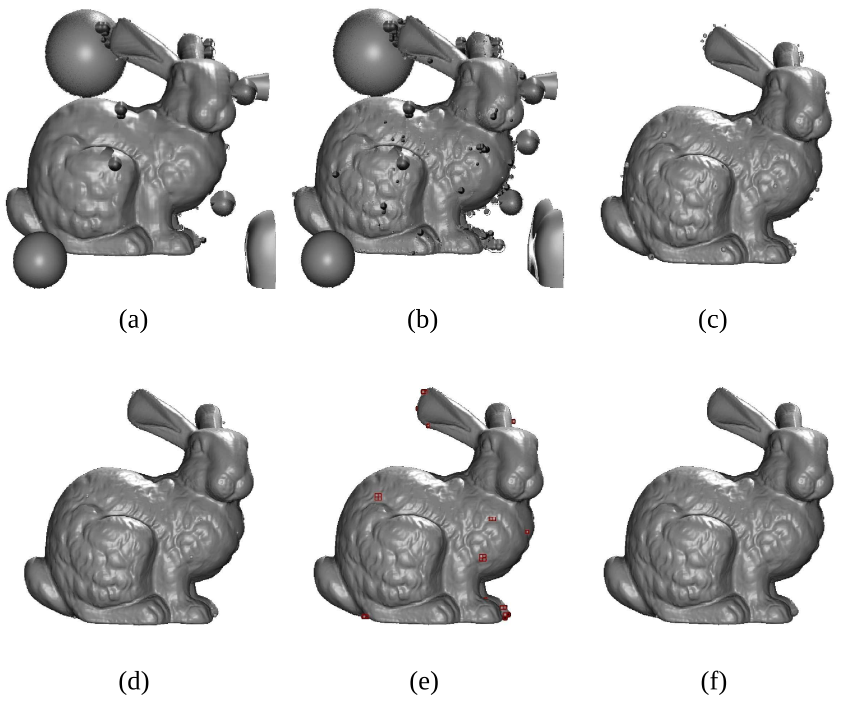


Figure 4.18: Function editing: (a)-(b) method by Ohtake *et al.* with its default parameters and with suggested parameters. (c) reconstruction using the proposed method without coverage domain; (d) reconstruction with coverage domain; (e) user selected imperfections; (f) function changed in order to eliminate imperfections. Comparisons against other surface reconstructions can be found in the work by Kazhdan and Hoppe.

criterion. One can notice the presence of several artifacts on the surface and some extra sheets, that were almost eliminated with the use of the coverage test (Figure 4.18d). In Figure 4.18e, a selection of blocks was defined, for which a constant function was set with positive values (0.002), in order to eliminate some of the surface flaws. Finally, Figure 4.18f depicts the bunny after the function editing.

The models reconstructed using Ohtake's technique presented a series of extra sheets and surface artifacts. In the work by Kazhdan and Hoppe [79], similar problems were presented for other techniques. Nevertheless, the differences with the results presented by Kazhdan and Hoppe are due to polygonization technique they employed, which generated the mesh for only one connected component. This was responsible for hiding most of the spurious sheets. The situation illustrated by Figure 4.18, shows that the set of solutions proposed in this paper considerably enhances the robustness of the reconstructions, given the fact that, even without the coverage criterion and in the presence of noise, the method was able to minimize the number of defects.

Also concerning function editing, in Figure 4.19 another example is presented,

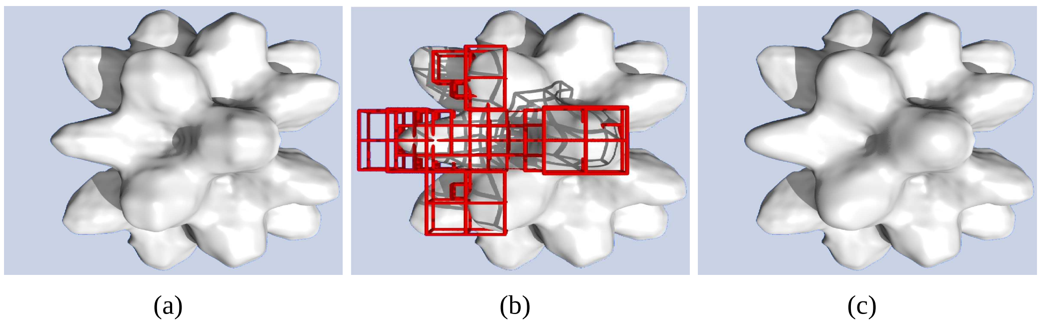


Figure 4.19: Enhancing the model using function editing: (a) model without high order approximations (due to configuration), (b) selected blocks for function changing and (c) final result.

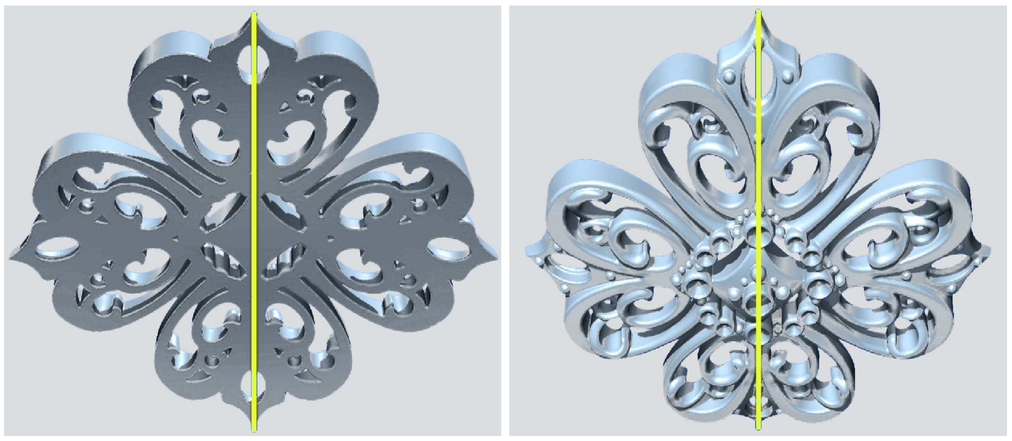


Figure 4.20: The Filigree model (514K points). The back and front views of the model are shown, in which the left half was generated with Ohtake's technique and the right half was generated with the method proposed.

in which planar functions, employed due to a user-defined configuration (Figure 4.19b), were replaced by second degree polynomials. Differently from the previous example, in this one we used the function editing to enhance the quality of the function and not to remove defects. The comparison between Figures 4.19a and 4.19c illustrates the gain in reconstruction quality.

It is important to mention that the method proposed here performs slightly worse in terms of computation time than the method by Ohtake *et al.* (about 5% slower). One major difference between the methods is that, in the approach presented, the evaluation is decoupled from the function approximation, in the sense that the whole function is built before the first evaluation is made. This fact means that for coarse grids or small ray tracer viewports, Ohtake's method tends to be faster, but as soon as the number of required function evaluation increases,

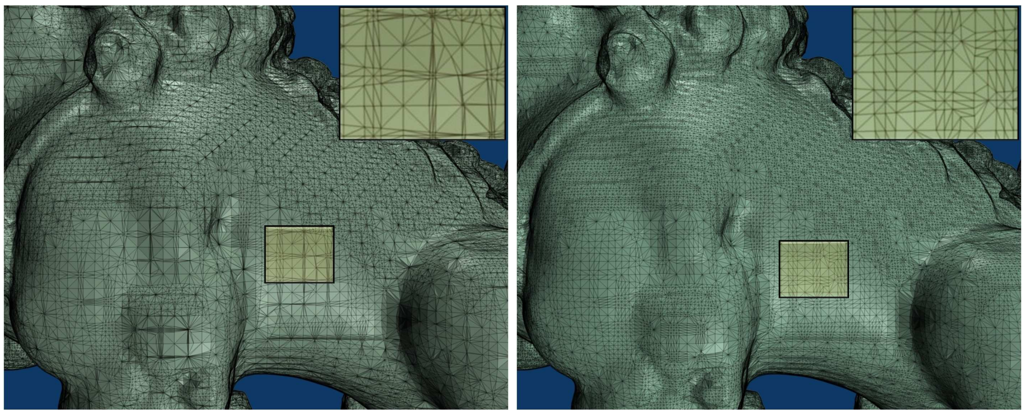


Figure 4.21: Comparing the iso-mesh produced from J_A^1 (left) against the iso-mesh obtained from J_A^1 with displacement (right). See color plates.

the methods perform similar in terms of processing time.

Another substantial difference between the methods is the function behavior in regions away from the zero-set. The proposed method presents a bounded maximum gradient magnitude for the whole domain due to the robustness criteria applied during the approximation phase. For instance, for the model presented in Figure 4.20, the ray-tracer algorithm we employed [128] showed a maximum gradient magnitude of 1.9 for the method presented and of 10^{10} for Ohtake's.

In order to illustrate the results of the mesh displacement technique, the Chinese dragon model (665K points) was reconstructed using a maximum refinement level 7. Figure 4.21 shows, on the left, the original mesh and, on the right, the model generated with mesh displacement. Both meshes are composed of 535 605 triangles and the time taken for polygonizing the models was 39.433s and 202.9s for the normal and the displaced J_A^1 version, respectively. This slow-down was expected for the displaced version, since it requires more evaluations of the function to compute the approximation of the gradient; but even so, this extra cost is constant and does not affect the complexity of the algorithm.

To assess the improvement of the triangles caused by the displacement, the aspect ratio measure $\alpha_\Delta = \frac{\sqrt{3}e_{max}P}{12A}$ was used, where e_{max} is the largest edge, P is the perimeter and A is the area of the triangle. Notice that the best aspect ratio is 1.0 (equilateral triangle).

For the normal mesh, the average aspect ratio was 5.55 and the standard deviation was 128.09, whereas for the enhanced mesh, the average was 1.68 and the standard deviation was 0.61. This result confirms the effectiveness of the technique because it was able not only to enhance the average the quality of the triangles but also to decrease the variation of the aspect ratio, which means that the whole mesh presents a reasonable overall aspect ratio.

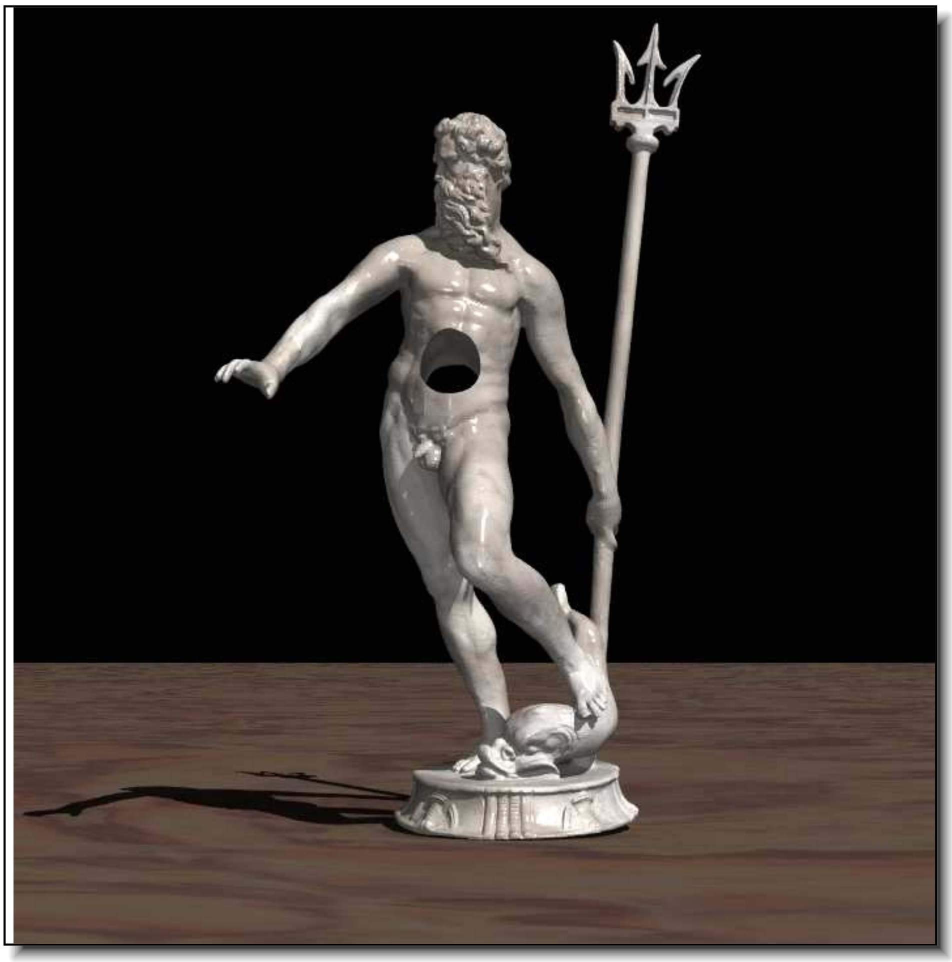


Figure 4.22: A CSG difference operation involving the Neptune model and a cylinder (see color plates).

Finally, in Figure 4.22, a CSG difference operation between the Neptune model and a cylinder is demonstrated. The support for such operations is an important advantage of implicit surface definitions.

4.5 GPU-based Rendering of Meshless Surfaces

Meshless surface representations can be directly visualized by using meshless surface rendering algorithms instead of generating a surface mesh. There exist a wide range of algorithms but the focus of this sections is on surface rendering algorithms for surface representations based on projection operators and on implicit functions. Specifically, it is shown how ray-tracing can be implemented on the GPU to render such surface representations.

4.5.1 Rendering surfaces based on projection operators

A GPU algorithm for ray-tracing surface representations based on projection operators, which is considerably faster than CPU implementations [150; 151], is presented here. This GPU implementation is based on the ray-tracing algorithms for moving least-squares surfaces proposed by Adamson and Alexa [3] and is tuned for the moving least-squares projection operator proposed by Levin [98] but can be naively modified to support other projection operators.

The process is described below where each step represents a rendering pass. As in the work by Adamson and Alexa, support balls are defined around the sample points, which will be rendered during the steps 'intersection', 'form covariance matrix', and 'form system for polynomial fitting'. During the steps 'find normals' and 'solve linear system and find projection' a quad covering the entire viewport is rendered to generate a fragment per ray. The step 'initial approximation' is performed only once as a pre-processing step.

Initial approximation. In this render pass, local polynomial approximations for each sample point $\mathbf{x}_i$ are calculated. For this, a single quad is rendered to generate a fragment per sample point. Each fragment calculates the corresponding local polynomial following Alexa's method to project a point onto the surface[1], for which, given the point $\mathbf{r}$ to project, the approximate tangent plane is obtained by solving

$$\min_{\mathbf{n},t} \sum_{i=1}^{N} \langle \mathbf{x}_i - (\mathbf{r} + t\mathbf{n}), \mathbf{n} \rangle^2 \omega_{MLS}(\mathbf{x}_i, \mathbf{r} + t\mathbf{n}).$$

The local polynomial fitting remains as described in Section 4.1 for Levin's definition. Since the point to be projected is the sample point itself, and is thus near the moving least-squares surface, $t = 0$ is assumed and the normal $\mathbf{n}$ is calculated using covariance analysis. Once $\mathbf{n}$ is defined, the polynomial is approximated in the local coordinate system (with $\mathbf{x}_i$ as the origin and $\mathbf{n}$ as one of the vectors of the orthonormal basis), with $\mathbf{x}_i$ as $\mathbf{r}$ and using neighborhood information prestored in a 3D texture to obtain the neighbors of the sample point $\mathbf{x}_i$. The result (coefficients) is rendered to a 32-bits per-channel float texture for further use.

Intersection. The nearest intersection of each ray with the local polynomials stored at the sample points defines the first approximation of the intersection of the ray with the point set surface. To find this intersection, viewport-aligned discs with radius ρ are rendered as shown in Figure 4.23 (as a 2D example). Each fragment belonging to a disc calculates the intersection of the ray that passes through it with the polynomial stored at the respective sample point (top-right zoom in the figure). For this, the ray is transformed into the local coordinate system, defined

[1]Note, however, that Alexa's projection procedure does not generate points on the surface as noted by Amenta and Kil [11].

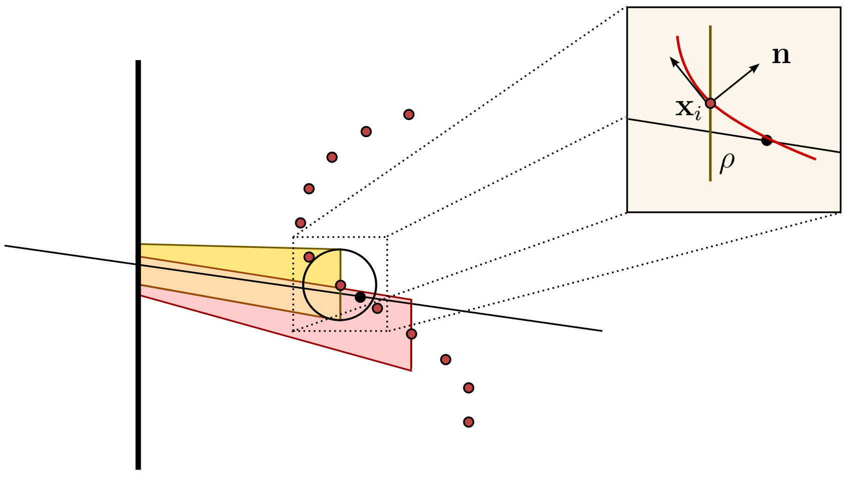


Figure 4.23: Calculating the intersection of the ray with the local approximation stored in each sample point (see color plates).

by $\mathbf{x}_i$ and $\mathbf{n}$ in the 2D example, where $\mathbf{n}$ is the normal calculated in the first step of the algorithm. If there is no intersection or if the intersection is outside the ball, the fragment is killed. Thus, using depth tests the nearest intersection $\mathbf{r}$ of the ray with the local polynomials is obtained. Once $\mathbf{r}$ is determined and stored in a float texture, its projection on the moving least-squares surface is found. This is done in the four next steps.

Form covariance matrix. Since the point $\mathbf{r}$ found in the last step is assumed to be reasonably close to the moving least-squares surface, $t = 0$ is assumed and $\mathbf{n}$ is found using covariance analysis. For that, the covariance matrix must be formed using the nearest neighbors of the point $\mathbf{r}$. Since performing k-nearest neighbors spatial searches is expensive, a range query is performed instead by rendering discs with a radius ρ sufficiently large to influence the points in the neighborhood of each point, being $\rho = 2h$ a good estimate for homogeneously sampled point clouds, where, as before, h is the fill size.

Each fragment generated this way calculates its distance to the intersection point on the ray passing through it in order to ensure that it is in the neighborhood of the intersection. In Figure 4.23 the zoomed disc influences the intersection point on the ray since it is within a distance ρ, whilst the influence of the disc in the back is discarded by means of a kill instruction for the fragment through which the ray passes. Each fragment belonging to the disc corresponding to $\mathbf{x}_i$ that passes the proximity test calculates $(\mathbf{x}_i - \mathbf{r})(\mathbf{x}_i - \mathbf{r})^T \omega_{MLS}(\mathbf{x}_i - \mathbf{r})$. The results of the fragments in the neighborhood of $\mathbf{r}$ are accumulated using one to one blending to three 16-bit-per-channel float textures that hold the 3×3 matrix (since blending to a 32-bit-per-channel texture is prohibitively slow).

Find normals. In this step a single quad covering the viewport is rendered to generate a fragment per ray. Each fragment calculates the eigenvector associated to the smallest eigenvalue of the matrix obtained in the previous step, using a GPU implementation of the inverse power method. The result is written to a float texture.

Form system for polynomial fitting. Once the normal at each intersection point is found, we must calculate the polynomial approximation using WLS. For that, a linear system is formed whose solution will provide the coefficients of the polynomial. The process is similar to the one of the step 'form covariance matrix' with the difference that the value calculated by each fragment belonging to the disc corresponding to $\mathbf{x}_i$ is twofold, a 4×4 matrix $\omega_{MLS}(\mathbf{x}_i, \mathbf{r})\mathbf{a}\mathbf{a}^T$ and a vector $\omega_{MLS}(\mathbf{x}_i, \mathbf{r})\mathbf{a}$ of size 4, where $\mathbf{a} = [(\mathbf{x}_i - \mathbf{r})_x^2 \;\; (\mathbf{x}_i - \mathbf{r})_y^2 \;\; (\mathbf{x}_i - \mathbf{r})_x(\mathbf{x}_i - \mathbf{r})_y \;\; 1]^T$. These results are accumulated by means of blending to four float textures to be used as input for the next step.

Solve linear system and find projection. The linear system formed in the last step is solved in a further render pass by rendering a quad covering the viewport. Each fragment (ray) solves the respective linear system using conjugate gradient. Then, the intersection $\mathbf{r}$ is projected onto the polynomial. If the distance between $\mathbf{r}$ and its projection is smaller than a threshold, the intersection of the ray with the surface has been found to be $\mathbf{r}$. Otherwise, the intersection of the ray and the local approximation is calculated. If the intersection is inside the ball with its center in the original sample point $\mathbf{x}_i$ and with radius ρ, this intersection is written into a float texture in order to use it in the next iteration.

The next iteration starts in the step 'intersection' where, from the second iteration on, it is necessary to check if the result of the last iteration, *i.e.*, the result of 'solve linear system and find projection', is already the intersection with the moving least-squares surface, in which case no further processing is done in any of the following steps. Otherwise, if the result of the last iteration is a valid intersection point within the ball defined by $\mathbf{x}_i$ and ρ, the following steps are performed using this intersection as the new point $\mathbf{r}$. If not, the nearest intersection, after the current $\mathbf{r}$, between the local approximations and the ray is found by means of depth tests as in the first iteration, killing all fragments with depth less or equal the depth of the current $\mathbf{r}$.

As reported by Adamson and Alexa, two to three iterations are needed to find all intersecting points between the primary rays and the moving least-squares surface. It is important to note that in the case of calculating the intersection between the moving least-squares surface and a set of rays that are not consistently oriented as the primary rays (*e.g.* secondary rays) the search for the intersection in step 'intersection' must follow another strategy. A naive brute force scheme was used, checking all local polynomial approximations for each ray (fragment). This can be performed using nested loops and/or multiple rendering passes. Since in

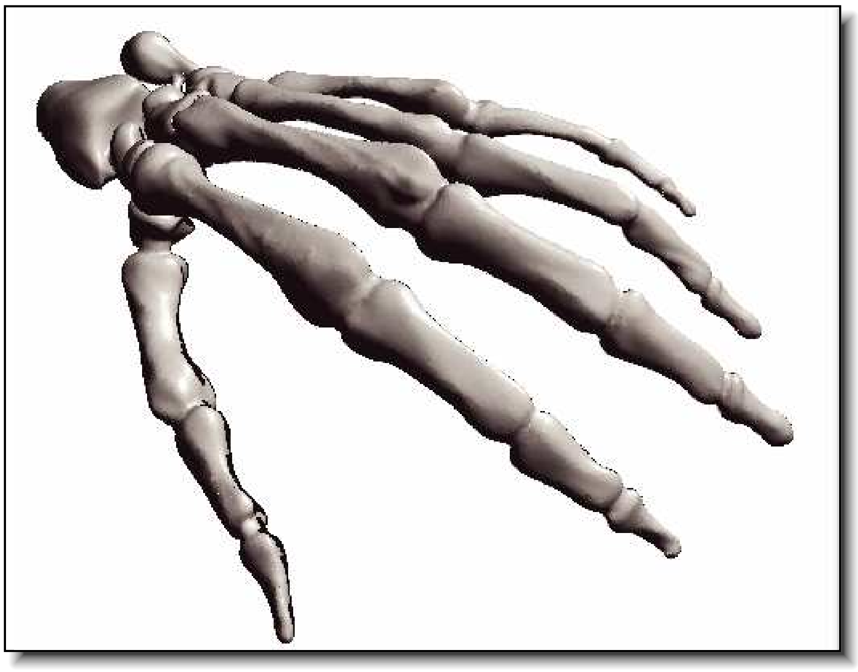


Figure 4.24: GPU-based ray-tracing of the moving least-square surfaces for the Skeleton Hand point cloud.

this case it is not possible to perform the range queries as described above, this first intersection can be used as a rough approximation of the intersection between the ray and the moving least-squares surface for secondary reflected, refracted, and shadow rays. Thus, for these secondary rays none of the four remaining steps is performed.

Ray-tracing a moving least-squares surface on the CPU could be prohibitively slow, whilst the proposed implementation achieved up to 6.25 fps for the Stanford Bunny (36K points), 4.54 fps for the Horse (48K) and 2.22 fps for the Skeleton Hand (109K) using the GPU implementation described with 2 iterations. In the case of 3 iterations the frame rate dropped to 4.16fps, 3.13 fps and 1.37 fps for the Bunny, the Horse and the Skeleton Hand respectively. However, 2 iterations suffice to generate a high-quality rendering as shown in Figure 4.24. The test were carried out on an Nvidia Geforce 8800 Ultra graphics card.

For the tests performed, a single reflection secondary ray and a depth of 2 were used. The use of secondary rays in the implementation described is currently limited by the lack of a proper data structure for performing range queries efficiently on the GPU. Therefore, the inclusion of such a data structure is of major importance and is addressed in the next section for rendering implicit surfaces.

The advantages of hardware-accelerated rendering of moving least-squares

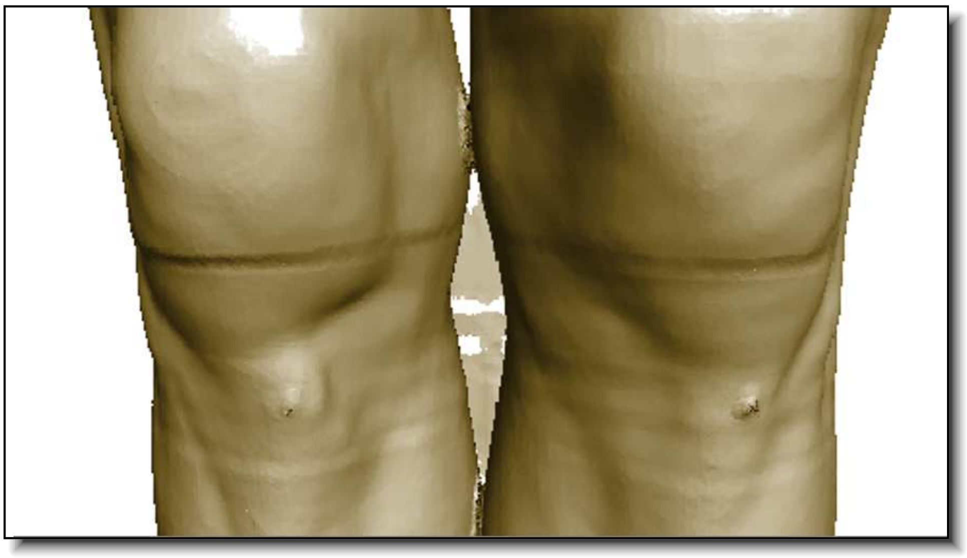


Figure 4.25: Point set surface for the vertices of an iso-surface mesh extracted from the Knee dataset with marching cubes.

surfaces could be exploited for accurate rendering of surfaces modeled as point clouds extracted from different sources, like medical data. For instance, the vertices of a surface mesh generated from MRI data by means of traditional methods, such as marching cubes or surface reconstruction techniques applied over the result of a segmentation algorithm, could be used as input to the ray-tracing algorithm. This will provide a smooth noise-free rendering of the iso-surface extracted. Figure 4.25 shows an example of this process for the Knee dataset.

4.5.2 Rendering implicit surfaces

Implicit surfaces can be rendered with the approach briefly described in Chapter 2. A GPU implementation of this approach is presented here. As in the case of hardware-accelerated rendering of surface definitions based on projection operators, the implementation presented here is based on the assumption that the surface is defined on a tubular region around $\mathcal{X}$ and this region can be approximated as the union of the enclosing spheres with radius ρ (see above) centered at the sample points.

The implementation for implicit surfaces is considerably easier than the implementation for projection operators. The algorithm, in this case, performs a single render pass. View-aligned discs with radius ρ centered at the sample points are rendered as described above. In this case, since no initial local polynomial surface approximation is available, each fragment computes the intersection points between the corresponding ray and the enclosing sphere of the sample point, whose disc generated the fragment. The implicit function is evaluated at both points and, in case a surface crossing is detected by comparing the signs of the resulting evaluations, the bisection method is used as depicted in Figure 2.2. To that end, the sample points in the neighborhoods of the evaluation points must be available. This can be accomplished using two different GPU data structures as explained in the following.

Gridded neighborhood. The first data structure is built by creating a three-dimensional Cartesian grid covering the domain of $\mathcal{X}$, with cell size equal to H. Each $\mathbf{x}_i$ is then assigned to the cell C in the grid containing it (in the Euclidean space). Additionally, $\mathbf{x}_i$ is added to the neighboring cells that intersect the support of the middle point of C. This support, in the implementation described here, is radial and, since Gaussian weighting functions were used for the implicit functions tested, all cells in a radius κH from the center are considered. The parameter κ must be large to ensure that all points that have a significant influence in the evaluation of the function at any evaluating point $\mathbf{x} \in C$. A good choice is $\kappa = 4$, since larger values compromise the performance. Once the grid containing the lists of points stored in each cell is created, the points in each cell are arranged consecutively in a two-dimensional array. Two such arrays are actually created, one containing the point positions and one containing the point normal vectors pre-calculated using covariance analysis as described throughout the chapter. This increases considerably the memory space required in favor of higher frame rates.

Three textures are then created with the grid and these arrays. The first texture (`grid`) is a three-dimensional texture with size equal to the resolution of the grid. Each texel in the texture contains the number of points stored in the corresponding cell as well as texture coordinates pointing to specific positions in the textures containing the arrays of positions and normal vectors (`positions` and `normals` respectively). To clarify what these last texture coordinates are, it is necessary to explain how the point positions and normal vectors are arranged in the two-dimensional arrays (which are directly mapped to two-dimensional texture). Since both arrays are arranged in the same way, the following discussion is valid for both the positions and the normal vectors arrays. The goal of the arrangement is to accelerate the fetching of the points stored in a cell. Thus, starting at the position (i, j) in the array containing the first point in the list stored in the cell C, the goal in turn is to require updating only the index i to access the following point in the list of points stored in C. To that end, the entire list of C, with size N_C, must be stored in N_C consecutive positions in the array (row-wise). This can be accomplished by fixing the width of the two-dimensional array as being equal to $\sqrt{M}$, where M is the sum of the sizes of all lists in the grid. Then, the lists in the gird are successively stored row-wise in the array until the number of points in a row is larger than the width. In this case, the next process continues with the next row in the two-dimensional array starting with the list that caused the over-

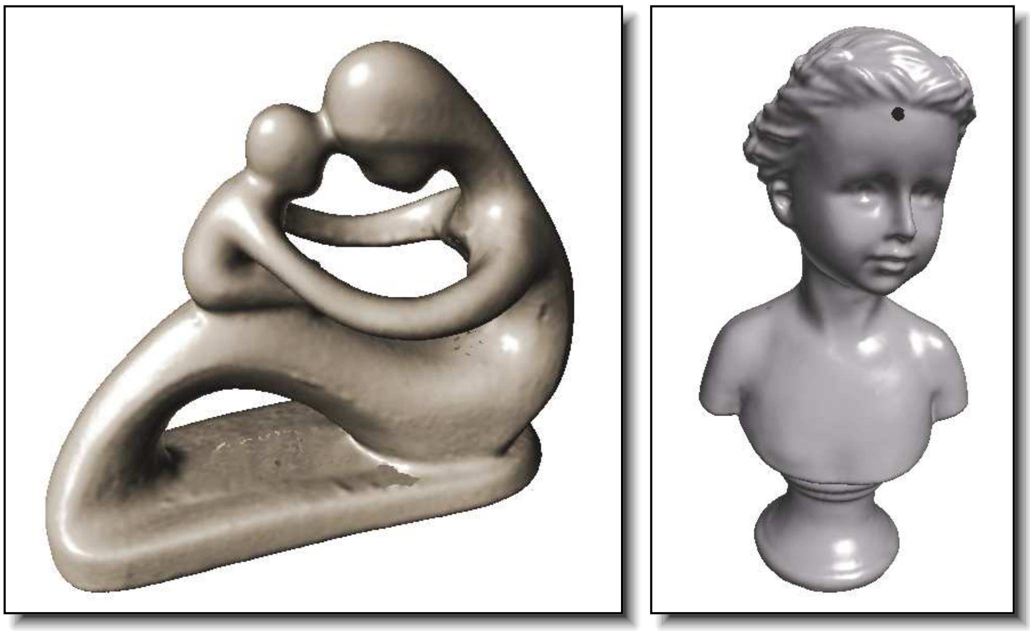


Figure 4.26: Renderings of the Fertility and Buste point sets using the GPU implementation described here with Kolluri's surface definition.

flow in the previous row (note that this entire list is stored in the new row and not partitioned among the two rows). Although this process causes further overhead in the storage requirements, it facilitates the access to the points in the fragment program.

During function evaluation in the fragment program, the cell $C_{\mathbf{x}}$ containing the evaluation point $\mathbf{x}$ is found and its texture coordinates in texture `grid` are computed using the resolution of the grid and the extent of the domain. The data in the corresponding cell is then fetched, which, as may be recalled, includes the size of the list of points stored in the cell as well as the texture coordinates of the position of the first point in the list in textures `positions` and `normals`. With this information, it is possible to access the texels containing the points in the neighborhood of $\mathbf{x}$ (this is, the points stored in $C_{\mathbf{x}}$).

Sample-centered neighborhood. The sample-centered data structure is similar to the gridded structure presented above. For each point $\mathbf{x}_i$, a list containing the points within a radius κH from $\mathbf{x}_i$ is computed. Thus, instead of creating a grid, a two-dimensional array of width and height equal to $\lceil \sqrt{N} \rceil$, where N is as before the number of points in $\mathcal{X}$, is used in order to generate enough array positions to correspond with all points in $\mathcal{X}$. Thus, the textures `positions` and `normals` are arranged in the same manner as before using the lists stored at the sample points and the texture coordinates to the first position of each list are stored in the corresponding position in the two-dimensional array replacing the grid, which is stored in texture `neighborhoods`.

In this case, given an evaluation point $\mathbf{x}$, the neighborhood of $\mathbf{x}$ is obtained by computing in the vertex program the texture coordinates (t_x, t_y) in texture `neighborhoods` where the data of the sample point corresponding to the currently processed disc is stored. For this, the index of the sample point is given as an element of the texture coordinates $\mathbf{t}_{CPU}$ passed by the application to the vertex processor for each disc. The resulting texture coordinates (t_x, t_y) are then passed to the fragment processor. Then, the fragment program uses (t_x, t_y) to fetch the number of points in the neighborhood and the texture coordinates in textures `positions` and `normals` where the first point in the neighborhood is stored. The entire list of neighboring points is then accessed as described above each time the implicit function must be evaluated. The new capabilities for working with arrays offered by the NVIDIA's G80 graphics cards was explored to fetch the positions and normal vectors only once, storing them in a global array, but this produced a dramatic performance decrease.

Independently from the data structure used, the list of points traversed for evaluating the implicit function contains points that may not lie inside the support of the weighting function centered at the evaluation point $\mathbf{x}$. Thus, in the fragment program, it is important to check for this condition when evaluating the function so as to discard those points in the list that do not belong to the support.

Both data structures delivered the same results in terms of surface approximation, as observed in Figure 4.26 where rendering of two point clouds using the data structures described above for Kolluri's [83] surface definitions are shown. Adamson and Alexa's [2] definition produced similar reconstruction and performance results. Although both data structures produce the same surface approximation, the sample-centered neighborhood out-performs the gridded neighborhood in terms of processing time. Using an NVIDIA Geforce 8800 Ultra graphics card, for the Fertility point set (107K points), gridded neighborhoods achieved 3.27 fps and sampled-centered neighborhoods 4.36 fps, whereas for the Buste point set (125K points), gridded neighborhoods delivered 1.28 fps and sample-centered neighborhoods 2.19 fps. Despite being faster, sample-centered neighborhoods are not adequate when following secondary rays in a ray-tracing implementation. In this case, the gridded neighborhoods better accommodate the requirement of finding the neighboring points for an arbitrary position on an arbitrary ray.

CHAPTER

5 MESHLESS SURFACES FROM VOLUMES

Volume visualization can benefit from meshless techniques that are currently being applied to surface data. A number of authors have explored this possibility in the last years. The most natural application for meshless surface modeling and rendering techniques in the context of volumetric data is the extraction and rendering of surfaces that are normally used to visualize volumetric data, such as isosurfaces and stream-surfaces.

Thus, in this chapter, results on meshless techniques developed for extracting surfaces from volumetric data are reported. In Section 5.2, a method for extracting moving least-squares surfaces from volumetric data is presented,which was developed in collaboration with João Paulo Gois from the Universidade de São Paulo. In Section 5.3, a method for the interactive extraction and rendering of stream-surfaces and path-surfaces is described. This method was developed in collaboration with Tobias Schafhitzel from the Universität Stuttgart, who must be credited for the hardware-accelerated technique used to generate a dense set of streamlines. Before detailing the techniques developed, related work is described.

5.1 Meshless Surface Extraction from Volume Data

Rendering volumetric data using meshless strategies has been addressed by different authors in the last years. Co *et al.* [34] proposed a method for isosurface rendering of volumetric data stored in Cartesian grids, named 'iso-splatting', which samples points in the domain and projects them onto the isosurface. The projected points are then rendered using splatting. The sampling is performed by finding the voxels that intersect the isosurface and adding its middle point to the list of samples. The sampled points are then projected onto the surface using either an exact or an approximate projection method. The projection of $\mathbf{x}$ is calculated by defining a ray $\mathbf{r} = \mathbf{x} + t\mathbf{d}$ and finding the root of

$$f_I = f(\mathbf{x} + t\mathbf{d}),$$

for t, where f_I is the isovalue and f is the tri-linear reconstruction of the scalar field. In practice, the direction $\mathbf{d}$ defining the ray is given by the vertices of the corresponding voxel that lies on the opposite side of the isosurface with respect to $\mathbf{x}$. This approach is slow since the roots of a cubic polynomial must be calculated. Therefore, the authors find an approximation by means of the Newton-Raphson method.

Co *et al.* [32] addressed the problem of generating isosurfaces from multiblock datasets with non-conformal cells. Their approach starts by generating an isosurface mesh with marching cubes. This mesh is then used to generate a set of sample points. The sample points are the input to a correction algorithm based on radial basis functions which melds the non-conformal surfaces obtained with the marching cubes algorithm. The correction algorithm uses a meshless radial basis functions interpolation of the isosurface and projects the sample points onto it. The interpolation is obtained by constructing a regular grid and using the middle point of each cell as center of the radial function. The projected points are then rendered using surface splatting.

The authors extended this approach to extract isosurfaces from large scattered datasets [35]. To that end, marching tetrahedra is used, instead of marching cubes, over a set of local tetrahedralizations which are computed so that their union covers the entire domain. This is achieved by constructing a regular grid covering the domain and storing the scattered points in the respective cell of this grid. Furthermore, a scattered point is added at the center of each void cell. Thus, the computations can be parallelized by dividing the regular grid into blocks, each of which is processed by a thread. The points in the block are used to construct a Delaunay tetrahedralization. As mentioned before, marching tetrahedra is used to obtain an isosurface mesh which is in turn used to perform the same meshless correction described above for multiblock datasets.

Livnat and Tricoche [107] proposed an hybrid view-dependent method for isocontouring based on points and triangles. Nested-grids are employed to traverse the domain and decide whether a triangle or a single point must be rendered. Starting at the root node, the algorithm prunes the current node in the traversal if the isovalue is outside the range stored in the node. In case the isovalue is contained in the node, and if the node is a leaf, geometry is generated. Otherwise the visibility of the children is determined and visible children are processed in visibility order. The geometry extraction step represents a node with a single point in case the projection of the cell covers a pixel or less. Otherwise, marching cubes is used to generate a local surface mesh.

Miriah *et al.* [116] address the problem of generating isosurfaces from data obtained from simulations that make use of the high-order finite element method, which is defined by basis functions in *reference space*, that give rise to a *world space* solution through a coordinate transformation, which does not necessarily has a closed-form inverse. Since methods such as marching cubes and ray-casting, which perform operations in world space, must compute an expensive nested root finding process, the authors address the problem with particle systems. A set of particles is thus distributed on the surface using geometric information from the world-space isosurface while performing the sampling in reference space. Given the set of input particle positions $\mathcal{X} = \{\mathbf{x}_1 \cdots, \mathbf{x}_N\}$ to be distributed across the surface and an implicit function $f(\mathbf{x})$, whose level set c is the desired isosurface, the positions $\mathbf{x}_i$; $i = 1, \cdots, N$, are iteratively refined as

$$\mathbf{x}_i \leftarrow \mathbf{x}_i - f(\mathbf{x}_i)\frac{f_{\mathbf{x}}(\mathbf{x}_i)}{f_{\mathbf{x}}^T(\mathbf{x}_i)f_{\mathbf{x}}(\mathbf{x}_i)}, \tag{5.1}$$

where $f_{\mathbf{x}}(\mathbf{x}_i)$ is the gradient of the implicit function at $\mathbf{x}_i$, until all particles lie within an error threshold ϵ_T of the surface. For each particle on the surface, a compact monotonically decreasing energy kernel E is associated. The energy E_i at a particle is then given by

$$E_i = \sum_{j=1,j\neq i}^{N} E_{ij} = \sum_{j=1,j\neq i}^{N} E\left(\frac{\|\mathbf{r}_{ij}\|}{\alpha}\right),$$

where $\mathbf{r}_{ij} = \mathbf{x}_i - \mathbf{x}_j$, and α defines the extent of the kernel such that when $|\mathbf{r}_{ij}| > \alpha$, $E_{ij} = 0$. Since using Euclidean distance fails to cull spatially close neighbors that lie on adjacent surfaces, the neighbors with normal vectors that are more than a 90-degree difference from the normal of the particle $\mathbf{x}_i$ are discarded.

The derivative of E_i with respect to the position of the particle is used to move the particle to a to a locally lower energy state

$$\mathbf{v}_i = -\frac{\partial E_i}{\partial \mathbf{x}_i} = -\sum_{j=1;j\neq i}^{N} \frac{\partial E_{ij}}{\partial \|\mathbf{r}_{ij}\|}\frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}.$$

The particle positions are then updated using the projection of the derivative of E_i onto the local tangent plane

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \left(I - \frac{f_{\mathbf{x}}(\mathbf{x}_i)f_{\mathbf{x}}^T(\mathbf{x}_i)}{f_{\mathbf{x}}^T(\mathbf{x}_i)f_{\mathbf{x}}(\mathbf{x}_i)}\right)\mathbf{v}_i,$$

where I is the identity matrix.

Since performing this operation can result in particles pushed off the surface, a re-projection using Equation 5.1 is necessary to ensure that the particles are within ϵ_T of the surface. This iterative two-steps process generates particles on the surface that are evenly distributed. To adapt the density of the particle sample to the details on the surface, the radius of the energy function can be scaled, for instance, according to the curvature. This mathematical framework is applied to high-order finite elements by means of expressions carefully formulated using Einstein notation in order to achieve world space adaptivity using reference space evaluations of the basis functions, mapping functions and their derivatives. This is done in order to avoid computing the inverse of the mapping function between reference space and world space.

5.2 Moving Least-squares Iso-surfaces

Moving least-squares surfaces proposed by Levin [98] have been further studied by Amenta and Kil [10; 11], who noted important properties about the domain of a moving least-squares surface and the behavior of the moving least-squares and weighted least-squares minimization strategies in the context of moving least-squares surfaces. Based on their observations, a novel technique to extract surfaces from volumetric data is proposed, inspired by the well-known 'predictor-corrector' principle. The method proposed is able to provide good approximations to the surfaces defined by a given feature in the volume, such as isosurfaces and surfaces located at regions of high gradient magnitude (HG-surfaces). This last class of surfaces is addressed since, as Kniss *et al.* [81] pointed out, although there is no mathematical prove, regions of interest are assumed to be located at regions of high gradient magnitude.

Mesh surfaces extracted from volumetric data have some inherent disadvantages, such as the need for defining the polygon characteristic size and the need for storing topological information. Also, important details may be omitted or coarse regions might be excessively detailed. Although more sophisticated methods were introduced in order to handle these problems [138; 139; 115], the mesh must be locally recomputed and refined, which is computationally expensive. Also, it is necessary to define an initial surface, implying that the user must know *a priori* some characteristics of the object in order to define a good initial approximation. The authors also mention the possibility of handling noisy data if sophisticated strategies of refining and displacement of vertices are used.

On the other hand, the method presented handles these problems naturally. Since it is based on local polynomial approximations, the precision of the model can be locally defined. Also, low frequency noise in the data is easily handled, due to the fact that the local approximations are computed by means of least-squares approaches. The method presented generates smooth surfaces avoiding the piecewise approximation of mesh-based methods. As proof of concept, the hardware-accelerated ray-caster presented in the previous chapter was extended to handle the surfaces defined here. This is done since, as stated by Adamson and Alexa [3], moving least-squares surfaces have clear advantages over other representations when ray-tracing is used, namely the locality of the computations, the possibility of defining a minimum feature size and the fact that the surface is smooth and a two-manifold. As seen in the previous chapter, beside the inherent implications of these three characteristics, the second advantage can be exploited when computing the intersection of the ray with the surface, whilst the last one turns CSG operations feasible.

As stated before, a strategy inspired by predictor-corrector methods, which make use of two numerical approaches to solve ordinary differential equations, is defined here for extracting the surfaces. The first approach is a ‘predictor’ which provides a first rough solution but requires only limited information. This solution is the input to the ‘corrector’ which then finds a final more accurate solution. These processes can be iterated in order to improve the solution obtained.

5.2.1 Computing MLS surfaces from volumetric data

The first step of the method is performed by solving a moving least-squares approximation problem to find an initial estimate for the projection of a given point using a carefully defined weight function that characterizes the surface. This makes it possible to deal with points that are relatively far from the surface. As discussed before, the weights traditionally used in moving least-squares minimization schemes are given by some monotone decreasing function of the distance from the point $\mathbf{r}$ to be projected to the point (voxel in this case) $\mathbf{x}_i$ in the input set $\mathcal{X}$. Here, However, in this step information on ‘how close a voxel is to a feature in the volume’ is used in order to weigh the voxels $\mathbf{x}_i$. The function $\omega_j;\ j = \{1, 2\}$ used to weigh the voxels $\mathbf{x}_i$ determines the feature that defines the surface and therefore the surface itself. Isosurfaces can be generated with the approach proposed by using

$$\omega_1(\mathbf{x}_i) = \exp\left(-\frac{|v - f(\mathbf{x}_i)|^2}{\rho_1^2}\right),$$

where v is the isovalue defining the isosurface, $f(\mathbf{x}_i)$ is the scalar value at $\mathbf{x}_i$ and ρ_1 is a scaling factor (also regarded as a smoothing factor). Also, assuming that regions of interest are located at regions of high gradient magnitude, a class of surfaces that depicts changes in the material properties can be obtained by using

$$\omega_2(\mathbf{x}_i) = \exp\left[-\frac{1}{\rho_2^2}\left(1 - \frac{\|\nabla f(\mathbf{x}_i)\|}{\max\{\|\nabla f(\mathbf{x}_i)\|\}}\right)^2\right],$$

where ρ_2 is also a scaling factor. With these weighting functions a local approximating plane is computed by finding $\mathbf{q}$ and $\mathbf{n} = \mathbf{n}(\mathbf{q});\ \|\mathbf{n}\| = 1$, so that $\mathbf{n}$ minimizes

$$e_{pISO}(\mathbf{q}, \mathbf{n}(\mathbf{q})) = \sum_{\mathbf{x}_i \in \mathcal{N}(\mathbf{r})} \langle \mathbf{n}, \mathbf{x}_i - \mathbf{q}\rangle^2 \omega_j(\mathbf{x}_i);\ j = 1, 2, \tag{5.2}$$

where $\mathcal{N}(\mathbf{r})$ is the set of neighbors of $\mathbf{r}$, $\mathbf{n}$ is in the direction of the line through $\mathbf{r}$ and $\mathbf{q}$, and the directional derivative of $J_{pISO}(\mathbf{q}) = e_{pISO}(\mathbf{q}, \mathbf{n}(\mathbf{q}))$ in the direction of $\mathbf{n}(\mathbf{q})$, evaluated at $\mathbf{q}$ is zero, *i.e.*, $\partial_{\mathbf{n}(\mathbf{q})} J_{pISO}(\mathbf{q}) = 0$.

After the minimization a local coordinate system is defined by the plane $H(\mathbf{n}, \mathbf{q})$. On this local coordinate system, weighted least-squares is used to find a bivariate polynomial $g(\eta, \zeta)$ that locally approximates the surface using as weighting function $\omega_{\{1,2\}}$. Defining $\mathbf{p}$ as the projection of $\mathbf{q}$ on the fitted polynomial $g(\eta, \zeta)$, the corrector scheme starts by computing a second approximating plane by finding $\mathbf{x}$ and $\mathbf{a} = \mathbf{a}(\mathbf{x})$; $\|\mathbf{a}\| = 1$, so that $\mathbf{a}$ minimizes

$$e_{cISO}(\mathbf{x}, \mathbf{a}) = \sum_{i=1}^{N} \langle \mathbf{a}, \mathbf{x}_i - \mathbf{x}) \rangle^2 \Theta(\mathbf{x}_i) \tag{5.3}$$

where $\Theta(\mathbf{x}_i) = \omega_{\{1,2\}}(\mathbf{x}_i)\omega_{MLS}(\mathbf{x}_i, \mathbf{x})$, $\mathbf{a}$ is in the direction of the line through $\mathbf{p}$ and $\mathbf{x}$, and the directional derivative of $J_{cISO}(\mathbf{x}) = e_{cISO}(\mathbf{x}, \mathbf{a}(\mathbf{x}))$ in the direction of $\mathbf{a}(\mathbf{x})$, evaluated at $\mathbf{x}$ is zero, *i.e.*, $\partial_{\mathbf{a}(\mathbf{x})} J_{cISO}(\mathbf{x}) = 0$. Recall that $\omega_{MLS}(\mathbf{p}, \mathbf{q}) \equiv w(\|\mathbf{p} - \mathbf{q}\|)$, where w is a monotonically decreasing function.

Then, as in the predictor step, a local coordinate system is defined on the plane $H(\mathbf{a}, \mathbf{x})$ and a polynomial approximation is computed using weighted least-squares with the weighting function Θ instead of $\omega_{\{1,2\}}$. The projection γ of $\mathbf{x}$ on this polynomial fitting is the final projection of $\mathbf{r}$ on the moving least-squares surface. The resulting projected points can be input to any point-based rendering method, such as EWA surface splatting [129]. In the following, however, the focus is on describing a modified version of the ray-casting engine for surface definitions based on projection operators, to accommodate moving least-squares surfaces extracted from volumetric data.

5.2.2 Hardware-accelerated MLS Iso-surfaces and HG-surfaces

To interactively render a moving least-squares surface directly from the volumetric data, viewport-aligned slices clipped with the bounding box of the volume are rendered, separated from each other by a distance of $\rho = kh$ (in the direction of the view vector), where $0.5 < k < 1$ and h is the smoothing parameter used in ω_{MLS} during the corrector step. The idea behind this operation is that, since the minimal feature size of the moving least-squares surface must be greater than h, by taking steps smaller than h it is ensured that the intersection between each ray and the surface will be found.

In order to reduce the computation time, the per-voxel information to be used is pre-computed and those fragments for which this information is smaller than a pre-defined threshold are discarded. For the case of isosurfaces this data is $|v - f(\mathbf{x}_i)|$ and for the surfaces located in regions of high gradient magnitude it is $\|\nabla f(\mathbf{x}_i)\|$. This threshold must be low enough to ensure that a sufficient number of fragments is used for the rest of the process.

For each fragment generated that passes the above mentioned test, the following computations are performed in a single rendering pass. The predictor step starts by minimizing Equation 5.2 defining the point $\mathbf{r}$ to be projected as the position of the fragment in space. This minimization is performed by means of an iterative process in which $\mathbf{q}$ and $\mathbf{n}$ are updated in each iteration until the change

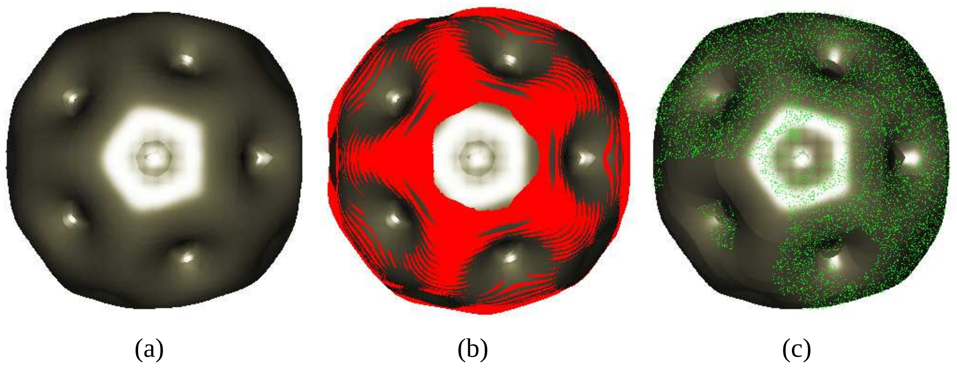


Figure 5.1: The Bucky Ball dataset. (a) The final result of applying the predictor-corrector method. (b) The points projected by the predictor at a distance greater than a pre-defined threshold are shown in red. (c) The output points from the predictor projected by the corrector at a distance greater than the threshold are shown in green (see color plates).

in $\mathbf{q}$ falls below a given threshold. The process starts by setting $\mathbf{q} = \mathbf{r}$. In each iteration, $\mathbf{q}$ is first fixed and covariance analysis is used to obtain the normal vector $\mathbf{n}$. To that end, the 3×3 covariance matrix

$$C_1(\mathbf{q}) = \sum_{i=1}^{N} (\mathbf{x}_i - \mathbf{q}) \otimes (\mathbf{x}_i - \mathbf{q}) \omega_{1,2}(\mathbf{x}_i)$$

is calculated and the eigenvector associated with the smallest eigenvalue of the matrix is computed using the inverse power method. This eigenvector gives us the normal vector $\mathbf{n}$. Then, $\mathbf{n}$ is fixed and Equation 5.2 is minimized by finding t, so that $\mathbf{q} + t\mathbf{n}$, where $\mathbf{q}$ is the current solution. Then we set $q \leftarrow \mathbf{q} + t\mathbf{n}$ and the next iteration starts. Finding t is straightforward since by fixing $\mathbf{n}$ the minimization of Equation 5.2 becomes a linear univariate minimization problem. Note that the points $\mathbf{x}_i$ that have a significant influence for these computations are the neighboring voxels of $\mathbf{q}$. Thus, no spatial search is required during the whole projection process.

Once $\mathbf{n}$ and $\mathbf{q}$ are found, a polynomial approximation to the surface is calculated in a local coordinate system defined over the plane $H(\mathbf{n}, \mathbf{q})$ using weighted least-squares and weighting the points $\mathbf{x}_i$ in the neighborhood of $\mathbf{q}$ with $\omega_{1,2}(\mathbf{x}_i)$. To exploit the capabilities of the GPU to handle vector operations for vectors of size 4, the polynomial

$$g(\eta, \zeta) = A\eta^2 + B\zeta^2 + C\eta\zeta + D$$

is used for the local approximation (note that (η, ζ) is in the local coordinate system). Therefore, the matrix of the linear system to be solved is of size 4×4 and

	Size (voxels)	Predictor	Predictor-Corrector
Cadaver Head	$256^2 \times 154$	2.92	0.14
Engine	$256^2 \times 110$	5.88	0.27
Fuel	64^3	9.26	0.88
Bucky	32^3	50.10	2.60

Table 5.1: Performance in frames per second for the moving least-squares surface extraction from volumetric data method.

thus easily handled in the shader. The projection of $\mathbf{q}$ on the local approximation gives us the point $\mathbf{p}$ which is used as input to the corrector step.

In the corrector step, the minimization of Equation 5.3 is performed in the same way as in the predictor step. The main difference is that when $\mathbf{a}$ is fixed to find $\mathbf{x}$, the minimization of Equation 5.3 remains non-linear. Thus, the *Brent with derivative* method was implemented to solve this problem. Also, the covariance matrix used for finding $\mathbf{a}$ is in this case given by

$$C_2(\mathbf{x}) = \sum_{i=1}^{N} (\mathbf{x}_i - \mathbf{x}) \otimes (\mathbf{x}_i - \mathbf{x}) \Theta(\mathbf{x}_i).$$

As before, once $\mathbf{x}$ and $\mathbf{a}$ are found, a local system is defined and weighted least-squares is used to compute a local approximating, this time using the weighting function Θ. The projection γ of $\mathbf{x}$ on this polynomial gives us the projection of the fragment's position $\mathbf{r}$ on the approximate surface.

Then, the ray-casting algorithm continues. If the distance between γ and $\mathbf{r}$ is less than a pre-defined error, $\mathbf{r}$ is the intersection of the ray with the approximate surface. Otherwise, as described in the last chapter, the intersection between the polynomial and the ray is found. If the intersection is within a region of confidence defined by a ball with radius ρ and center $\mathbf{r}$, the projection process is started again definding the intersection found as the new $\mathbf{r}$. This process is repeated until the distance between the projection and $\mathbf{r}$ is less than the error, or the intersection is outside the ball. In the last case the fragment is killed, which, since depth tests are used, simulates the jump to the next ball used by Adamson and Alexa.

Rendering and performance results of the methods proposed are presented in the following. All tests were carried out on a standard PC with a 3.4 GHz processor, 2GB of RAM and an NVIDIA Geforce 8800 Ultra graphics card. The size of the viewport used for the performance measurements was 512^2.

In Table 5.1, the results obtained for the extraction of surfaces from volumetric data are presented, performed using only the predictor step and the extraction performed using one iteration of the predictor and one iteration of the corrector steps. As can be noticed in the table, the corrector step adds a significant overhead

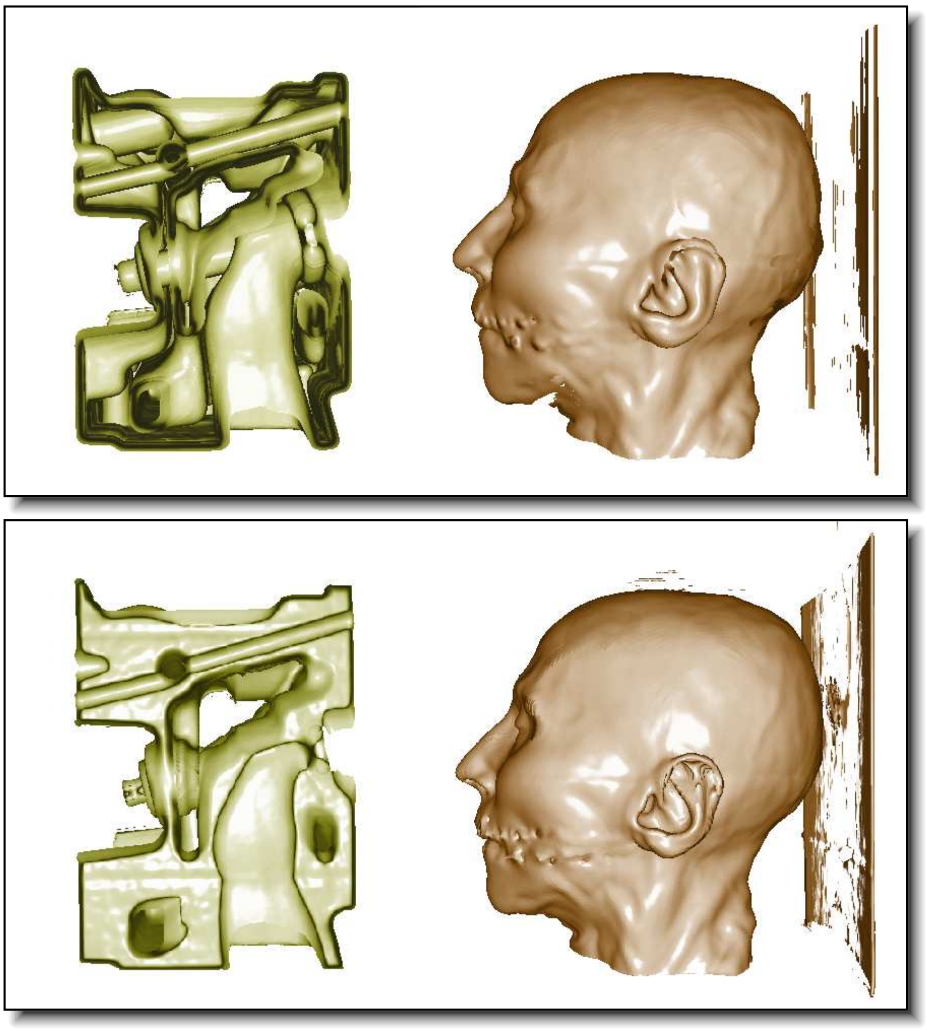


Figure 5.2: Moving least-squares surfaces extracted for the Engine and the Cadaver Head datasets using the gradient magnitude (top) and isovalues (bottom).

to the processing time. Although this step improves the accuracy of the result, for interactive applications where precision is not important, the predictor suffices to generate an already good approximation to the moving least-squares surface. This fact is depicted in Figure 5.1 where the effect of the predictor and the corrector steps on the input points (fragments) is shown. The predictor step projects a significant percentage of the points at a distance greater than a pre-defined threshold, set to test this effect. On the other hand, although the effect of the corrector step over the points already projected by the predictor is reduced to a small amount of points, this further projection could be important for applications where precision is the main concern.

The results obtained are promising considering the complexity of the computations involved. Although the implementation for extracting moving least-squares surfaces from volumetric data is not interactive for the predictor-corrector case, the processing time is considerably low in relation the large amount of fragments projected. Also, the renderings are of good quality as shown in Figure 5.2.

5.3 Point-based Stream Surfaces

Stream surfaces are a direct extension of streamlines, that is, surfaces that are everywhere tangent to the vector field. They are effective in simultaneously displaying various kinds of information of a flow, such as flow direction, and torsion of a vector field as well as in conveying vortex structure [55]. Despite these advantages, stream surfaces are not common in flow visualization. Such a lack of popularity may be due to the fact that stream surfaces require more advanced algorithms than streamlines; interactive visualization is not as easy to achieve as for streamlines; wide stream surfaces lack internal visual structure, leading to possible perception problems; and stream surfaces have been traditionally restricted to steady flow. These issues are addressed here by devising a new point-based algorithm for stream surface construction and rendering. Thereby, an expensive triangulation of the stream surface is avoided. Particle tracing starts at a curve of seed points and results in a collection of particles that represent the stream surface. More specifically, the issues mentioned above are addressed by developing a point-based computation of stream surfaces that maintains an even density of particles on the surface and by rendering the points by means of splatting. An extension to path surfaces of unsteady flows and the combination with texture-based flow visualization on stream surfaces and path surfaces to show inner flow structure on those surfaces are also described. Furthermore, it is shown how these algorithms can be mapped to efficient GPU implementations. The visualization approach makes it possible to interactively generate and render stream surfaces and path surfaces, even while seed curves are modified by the user or time-dependent vector fields are streamed to the GPU. Figure 5.3 illustrates an example of stream surfaces generated by the algorithm presented.

While the concept of a stream surface is straightforward, its implementation is more challenging than for streamlines because a consistent surface structure needs to be maintained. Hultquist [73] describes an algorithm that geometrically constructs a stream surface based on streamline particle tracing. In particular, his algorithm takes into account the stretching and compression of nearby streamlines in regions of high absolute flow divergence. Garth *et al.* [55] show how Hultquist's algorithm can be improved in order to obtain higher accuracy in areas of intricate flow. An alternative computation is based on implicit stream surfaces [157], which however cover only a subclass of stream surfaces. A related line of research ad-

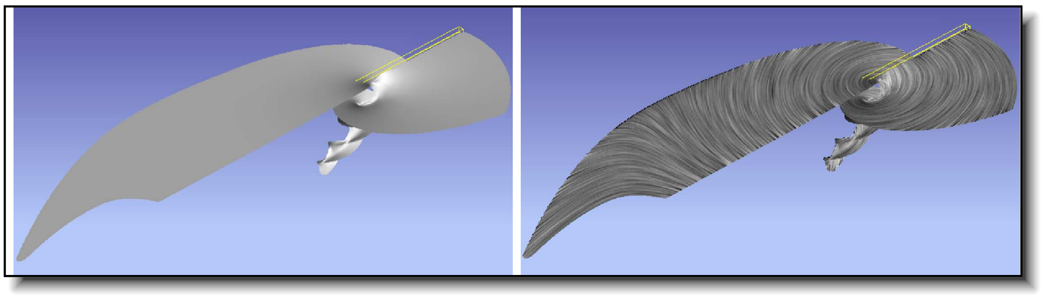


Figure 5.3: Visualization of the flow field of a tornado with: (left) a point-based stream surface; (right) the combination of a stream surface and texture-based flow visualization to show the vector field within the surface. Each stream surface is seeded along a straight line in the center of the respective image (see color plates).

dresses the issue of how stream surfaces are displayed effectively, for example, they can be chosen according to principal stream surfaces [27], rendered at several depths by using ray casting [53], or visualized through surface particles to reduce occlusion [156]. Previous methods are restricted to stream surfaces, to steady flow or instantaneous vector fields of unsteady flow, whereas the approach proposed is designed for steady and unsteady flow alike.

The approach proposed here adopts line integral convolution (LIC) [26], extended to tangential flow on curved surfaces. While several methods exist for texture-based flow visualization on surfaces [93], a hybrid object/image space LIC method [166] is used because it can process vector field data extracted by point-based rendering. The object/image space LIC is similar to texture advection in image space [94; 159], but achieves better filter quality and guarantees temporal coherence under camera motion. The basic visualization strategy described here resembles recent work by Laramee *et al.* [92], which combines mesh-based stream surfaces with texture advection for an improved visualization of steady flow: by construction, a vector field carved out on a stream surface is always tangential to the surface; therefore, a projection of a 3D vector field onto a surface is avoided. In addition, the visualization method proposed is designed for steady and unsteady flow alike as mentioned before. Typically, the texture-based visualization of unsteady flow leads to smeared-out texture patterns, as present in texture advection [77; 158] or UFLIC [142; 106; 102]. We show that the approach presented leads to clear line patterns that show a certain choice of path lines. More background information on flow visualization in general can be found in the book chapter by Weiskopf [165].

5.3.1 Streamlines and path-lines generation

Stream surfaces are surfaces that are everywhere tangent to a time-independent vector field. According to Hultquist [73], a stream surface can be represented as a 2D parametric surface embedded in a 3D flow. A natural choice of parameterization uses one parameter, $s \in [0,1]$, in order to label streamlines according to their respective seed points. Assuming a parameterized representation of the seed curve, we base s on that curve parameterization. The actual streamlines are computed by solving the ordinary differential equation for particle tracing,

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}(t), t), \tag{5.4}$$

where $\mathbf{x}$ is particle position and $\mathbf{v}$ is the vector field at time t. The seed points represent the initial values for the ordinary differential equation. Then, the second parameter of the stream surface is the time, $t \in [0, t_{\text{max}}]$, along the streamline integration. This choice of surface parameterization results in two meaningful classes of isoparameter curves: for constant s and varying t, streamlines are obtained; for constant t and varying s, time lines are obtained, which are advected images of the initial seed line.

For stream surfaces, a time-independent vector field $\mathbf{v}$ is assumed. However, the above construction is already designed for time-dependent vector fields. In this case, particle tracing leads to pathlines instead of streamlines, which in turn results in the construction of *path surfaces* instead of stream surfaces.

As the aim is to provide an interactive tool for the generation and visualization of those surfaces, the algorithm for generating streamsurfaces and path-surfaces is designed for a GPU implementation. The basic algorithm consists of three parts: the generation of the seed points, the integration of the particles along the given vector field, and insertion/removal events to maintain an evenly dense sampling of the surface by particles. The first part is executed once only at the beginning, whereas the second and third parts are repeatedly executed in an interleaved manner to incrementally construct the stream surface. A roughly even density of particles is maintained in order to obtain a good reconstruction of the surface during the point-based rendering process.

The data structures of the algorithm can be represented as two-dimensional textures. Texture `particles` stores the positions of the particles in the object space of the surface. The organization of the `particles` texture is rather simple: the number of rows stands for the number of particles, whereas the columns describe the number of integration steps. Actually, the number of rows has to be η times greater than the number of initial particles to allow for additional room for particles inserted during surface construction. Texture `states` is introduced to store additional data values and has the same size as texture `particles`. This texture contains indices to the left and right neighbors of the respective streamline. The vector field additionally is held in a three-dimensional texture.

In the first step of the algorithm, the seed points are generated. To generate the seed points, the user defines a seed curve by placing a straight line at a specific region of interest. Seeding is implemented by rendering only one column of texture `particles` (Figure 5.4a). The height of the quadrilateral used for rendering represents the number of initial particles. Similarly, the `states` texture is initialized with indices to streamline neighbors.

After initialization, the integration of particle traces is performed in the second step. First-order Euler integration is applied to solve Equation 5.4, but higher-order methods could be used as well. Particle tracing updates texture `particles` in a column-wise manner, where each column corresponds to a specific time. The previous position of a particle is obtained by a texture lookup using the texture coordinates that refer to the previous column. Then, the updated position is written to the current column. A ping-pong rendering scheme is used for updating the particle positions. The `states` texture is treated in the same manner to maintain consistent connectivity information.

The third step of the algorithm implements the insertion or removal of particles. This step relies on criteria [73] that decide whether a particle remains, needs to be added, or has to be removed. In addition, a stream surface may tear, for example in regions of very high divergence or when the flow hits an interior boundary. Let $\mathbf{x}_{i,t}$ be the position of the i-th particle at time t and $\theta(\mathbf{x}, \mathbf{y})$ be the distance between points $\mathbf{x}$ and $\mathbf{y}$. Then, a particle is inserted if

$$\theta(\mathbf{x}_{i,t}, \mathbf{x}_{i+1,t}) > \alpha\,\theta(\mathbf{x}_{i,0}, \mathbf{x}_{i+1,0}) \tag{5.5}$$

and

$$\theta(\mathbf{x}_{i,t}, \mathbf{x}_{i+1,t}) - \theta(\mathbf{x}_{i,t-1}, \mathbf{x}_{i+1,t-1}) < \beta\,\theta(\mathbf{x}_{i,t-1}, \mathbf{x}_{i,t}), \tag{5.6}$$

where α and β are usually set to 2. The first inequality tests if the current distance is larger than α times the initial distance between two adjacent particles. The second inequality guarantees that the distance between two neighbors does not grow more than β times faster than the distance between its previous and its current position. The surface tears if Equation 5.5 is true and Equation 5.6 is not met. A particle dies if the distance between two neighboring particles is too small, for example, when particles enter a convergent area of the flow. A particle is removed if the following conditions are fulfilled:

$$\left(\frac{\mathbf{x}_{i,t} - \mathbf{x}_{i-1,t}}{\|\mathbf{x}_{i,t} - \mathbf{x}_{i-1,t}\|}\right) \cdot \left(\frac{\mathbf{x}_{i+1,t} - \mathbf{x}_{i,t}}{\|\mathbf{x}_{i+1,t} - \mathbf{x}_{i,t}\|}\right) \approx 1 \tag{5.7}$$

and

$$\begin{array}{rlcl} & \theta(\mathbf{x}_{i,t}, \mathbf{x}_{i+1,t}) & < & \gamma\,\theta(\mathbf{x}_{i,0}, \mathbf{x}_{i+1,0}) \\ \wedge & \theta(\mathbf{x}_{i,t}, \mathbf{x}_{i-1,t}) & < & \gamma\,\theta(\mathbf{x}_{i,0}, \mathbf{x}_{i-1,0}), \end{array} \tag{5.8}$$

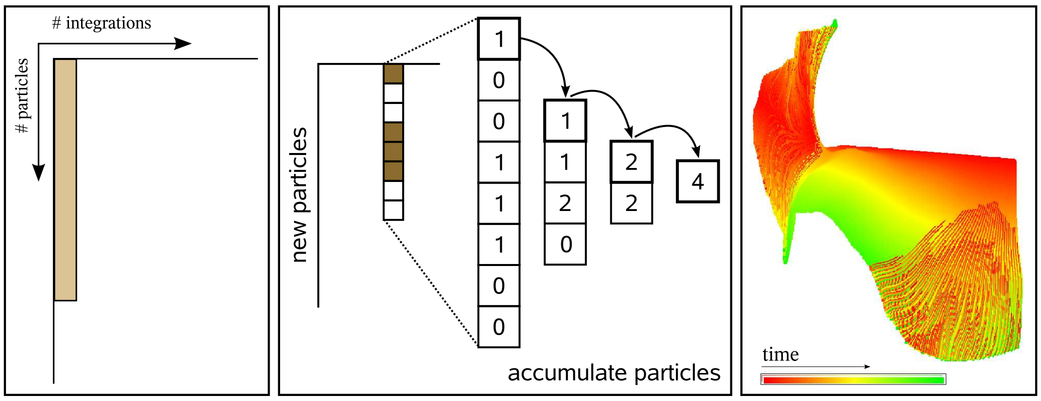
where γ should be less than 1. The dot product in Equation 5.7 tests for collinearity of the particle and its neighbors. If this is true, both distances from the particle to its neighbors are checked. Equation 5.8 defines that a particle needs to be removed if the distances to its neighbors are smaller than the distances at $t = 0$, scaled by γ.

The computation of the different criteria requires data from the local neighborhood of a particle. The temporal neighborhood (*i.e.*, access to previous time step) is intrinsically encoded in the `particles` texture because a row of that texture corresponds to different time steps of the same particle. The spatial neighborhood is explicitly stored in the `states` texture, which holds indices to the left and right neighbors.

Particle removal is implemented by marking "dead" particles in the texture `particles` so that they are not processed any further during particle tracing and surface rendering. By using render targets with floating point precision, no additional color channel is necessary. There exist at least two channels, containing the neighbors which cannot be negative. If the particle dies, one of these channels is used to store this additional information, by negating its current value. The implementation of particle insertion uses two additional textures that store intermediate results. The first one contains the positions of the new particles, and the other one contains the corresponding states. Both textures have the same height as the original `particles` and `states` textures. Each existing particle is tested with its right neighbor using Equations 5.5 and 5.6. If both inequalities are true, a new particle $\mathbf{x}'_{i,t}$ is created by linear interpolation between $\mathbf{x}_{i,t}$ and $\mathbf{x}_{i+1,t}$. Then, the particle position and connectivity are written to the additional textures. The neighbors are assigned to the new particle by using the coordinates of $\mathbf{x}_{i,t}$ as left and $\mathbf{x}_{i+1,t}$ as right neighbors.

The problem is that the intermediate textures may contain only a few particles that were actually inserted. In fact, most of the cells of those textures will contain inactive elements. Therefore, the intermediate textures must be condensed by removing all inactive particles and putting the active particles in a consecutive order. Such a reordering is rather complicated for a GPU implementation. The histogram pyramids proposed by Ziegler *et al.* [173] are adopted and slightly modified to fulfill this task. The main idea is to merge the positions of the new particles, which are distributed over the whole texture. Due to the fact that the particles' positions are updated column by column, the merging algorithm is restricted to a 1D domain. In fact, all new particles are stored in one column, in which either a texel is filled with a new particle or is empty.

A binary tree is built over this column by using a pyramid stack of 1D textures, where each level of the pyramid has at least half of the height of the previous level, representing one level of the binary tree. The finest level n represents the new particle itself. In the implementation presented, a flag is used to notify a texel if it contains a new particle ($\phi = 1$) or not ($\phi = 0$), which serves as basis for the binary tree generation. If rendering level $n - 1$ of the binary tree is the



(a) (b) (c)

Figure 5.4: Illustration of different steps of the algorithm: (a) during the initialization of the particles texture only one column is rendered (the height of the strip represents the number of initial particles) and (b) during creation of the binary tree the new particles build the highest level and the contents are summed up until the root contains the overall number of particles to be inserted. In (c) the lifetime of the individual particles is shown. The color gradient is defined from red (at $t = 0$) to green and illustrates the increasing lifetime. The areas with red lines at the left and bottom-right parts of the image show regions with many new streamlines (see color plates).

current one, for example, always two texels of level n are accumulated and stored into one texel of level $n - 1$. This is continued until the root level 0 is reached, which is represented by one texel containing the overall number of new particles (Figure 5.4b). The creation of the binary tree requires n render passes to build the tree levels in a bottom-up manner.

After the binary tree is created, the new particles are added to the actual texture `particles`. Here, the number of new particles is read back from graphics memory. Due to the small texture size (one texel) the texture read back does not affect the performance significantly. Then a quadrilateral is created with a height equal to the number of new particles. According to the histogram pyramid method, each texel rendered by the quadrilateral is numbered from 0 to $k - 1$, where k stands for the number of new particles. The adapted top-down traversal algorithm by Ziegler *et al.* [173] works as follows. Starting from the first level of the binary tree, the key value is compared with the entry of the current cell. If the key value is smaller than the cell value, the tree is traversed downward. If the key value is greater or equal, the value of the current cell is stored and the binary tree is traversed downward following the pointer of the successor. This is repeated until the algorithm reaches level n. Finally, the value of the current cell plus the number of predecessors gathered during the traversal is compared to the key value. If the key value is smaller, the new particle corresponding to the key value is found, otherwise the successor cell is used. By rendering the position and the corresponding states at the current fragment, the new particle is inserted containing all the necessary information for the growing of the stream surface. In fact, the left neighbor is the particle $\mathbf{x}_{i,t}$, which has created the new particle, and the right one is the old neighbor of $\mathbf{x}_{i,t}$, which was $\mathbf{x}_{i+1,t}$.

To restore the consistency of the particle system, the old particles need to be updated as well. When a new particle is created, its index in the `particles` texture is yet unknown because of the particle merging mechanism. To build the connectivity information, the binary tree is used to obtain the relation of the new particles to their old predecessors. Now, the tree is traversed bottom-up, from the leaves to the root. The particle $\mathbf{x}'_{i,t}$ stored in the temporary texture serves as basis of the traversal (note that it represents the leaf level n of the binary tree). While the binary tree is traversed upwards, each cell is tested if it builds the first or the second entry of a tuple. If it has a predecessor, the predecessor's value is accumulated before the algorithm ascends to the next tree level. When the root node is reached, the gathering algorithm stops and the accumulated value represents the number of predecessors. This information and the texture coordinates of the new particles are sufficient for reassigning the new neighbors. Please note that this algorithm has to be executed for all particles which have created a new one, as well as for their former right neighbors, because they also need to be informed about their new left neighbors.

The complete stream surface is constructed by applying the particle insertion/removal and particle integration processes several times. Figure 5.4c illustrates the lifetime of the individual particles. Color encodes an increasing lifetime of particles by red to green. New particles are identified as red areas surrounded by green streamlines. The maximum integration length is user-specified. The algorithm proposed is able to create stream surfaces and path surfaces alike.

For the subsequent LIC calculation the vector field is needed on the surface. The remaining three channels provided by the `particles` and `states` textures are used to store the attached vector of the flow. Regardless of whether steady or unsteady flow is visualized, only the vector used for the integration of the current time step is stored.

5.3.2 Point-based surface rendering

The fact that particles are added when divergence in the flow is present ensures a sufficiently dense sampling to cover the image space consistently. This way, it is only necessary to generate enough particles and render them as small point sprites in order to obtain a closed surface. However, in order to obtain lit surfaces, the normal vectors at each position on the surface must be determined. Therefore, the normal vectors are firstly estimated at the particles positions, which can be performed by means of covariance analysis, as in previous sections.

Given a point $\mathbf{q}$ in $\mathbb{R}^3$ and a set of points $\mathcal{X} = \{\mathbf{x}_i, \ldots, \mathbf{x}_N\}$ on the surface (the particle positions), the 3×3 weighted covariance matrix $\mathbf{C}$ is given by

$$\mathbf{C}(\mathbf{q}) = \sum_{\mathbf{p}_i \in \mathcal{N}(\mathbf{q})} (\mathbf{x}_i - \mathbf{q}) \otimes (\mathbf{x}_i - \mathbf{q}) \omega_{COV}(\mathbf{x}_i, \mathbf{q}), \tag{5.9}$$

where $\mathcal{N}(\mathbf{q}) \subsetneq \mathcal{X}$ is the set of neighbors of $\mathbf{q}$ and ω_{COV} is a non-negative monotonically decreasing function. Note that in practice, as done in all implementations described in this thesis, only the neighborhood of $\mathbf{q}$ is used instead of the entire data set to reduce computational costs, since the locality of the definitions (though the compact weighting functions) allows it. Once the matrix $\mathbf{C}(\mathbf{q})$ is calculated, the normal vector at $\mathbf{q}$ is estimated as the eigenvector of $\mathbf{C}(\mathbf{q})$ corresponding to the smallest eigenvalue. To find this eigenvector the inverse power method is used.

Given the layout of the texture holding the particles, this processing can be performed in one render pass. For a given particle position $\mathbf{q}$, the set $\mathcal{N}(\mathbf{q})$ is defined as the $\mathbf{x}_i$ corresponding to the particles of the previous and next time steps in the neighboring streamlines. These particles positions can be accessed using the connectivity information stored in the `particles` and `states` textures. Then, the computation of $\mathbf{C}(\mathbf{q})$ is straightforward and can be implemented in a single fragment shader, together with the inverse power method to obtain the normal vector at each particle position. This process is performed by rendering a single quadrilateral of the same size as the particles texture. The input to the fragment program are the textures `particles` and `states`. The former is used to fetch the particles' positions and the latter to fetch the neighboring particles' texture coordinates. The results of this render pass are stored in the texture `normals`.

Once the estimated normals are available, three further render passes are performed. The final result of this process is stored in three further textures, namely, the `intersections` texture with the intersection points on the surface; the `lit_surface` texture, which holds the projected and lit surface; and texture `vectors` with the interpolated vector of the flow at each position on the projected surface.

The process is started, for the first rendering pass, by rendering a quadrilateral for each particle centered at the particle position and perpendicular to the normal vector corresponding to the particle. The quadrilaterals are trimmed in the fragment program by means of clipping operations to obtain discs of radius ρ, where $0.5h < b < h$, where h is, as before, the smoothing factor. For this, in the vertex program, it is necessary to attach to each vertex of the disc its position and the position of the particle in object-space coordinates (since the normal vectors were also computed in object-space coordinates). The fragment program writes the position of the fragment (if not clipped) to the texture `intersections`.

This texture and the texture `normals` are the input to the second render pass. Discs centered at each particle position are rendered as in the previous pass. The vertex program in this case fetches the normal vector corresponding to the particle and attaches it to the vertex in addition to the positions of the vertex and the particle in object space. Each fragment thus generated writes the normal vector to the RGB channels and $\omega_{STR}(\|\mathbf{x}_j - \mathbf{q}_i\|, \|\mathbf{x}_j - \mathbf{z}_k\|)$ to the alpha channel of a first target texture `weighted_normals`. The $\mathbf{x}_j$ and $\mathbf{q}_i$ are the positions of the fragment and of the particle in object space respectively, and $\mathbf{z}_k$ is the intersection point from the texture `intersections`, stored in the texel corresponding to the fragment position in clipping space. The function ω_{STR} is defined

$$\omega_{STR}(r, s) = \exp\left(-\frac{r^2}{h^2} - \frac{s^2}{\mu^2 h^2}\right),$$

where the parameter μ controls the influence of the fragments that are behind the intersection point. This parameter is chosen to avoid the influence in the result of fragments that are not in the 2D neighborhood (surface) of the intersection point. Also, to obtain sharp intersections as in Figure 5.5, it is important to test if the texture coordinates (along the x-axis in the texture) corresponding to $\mathbf{q}_i$ are in the neighborhood of the texture coordinates of the particle corresponding to $\mathbf{z}_k$. This is done to ensure that only particles in the neighboring time steps are considered to calculate the normal and velocity vectors. A second texture `weighted_vectors` is attached to a second render target, where the fragment program writes, in the RGB channels, the velocity vector at the particle position and, in the alpha channel $\omega_{STR}(\|\mathbf{x}_j - \mathbf{q}_i\|, \|\mathbf{x}_j - \mathbf{z}_k\|)$. By using alpha blending, for each ray (pixel), the vectors $\sum_{\mathbf{x}_j} \omega_{STR}(\|\mathbf{x}_j - \mathbf{q}_i\|, \|\mathbf{x}_j - \mathbf{z}_k\|)\mathbf{n}_i$, and $\sum_{\mathbf{x}_j} \omega_{STR}(\|\mathbf{x}_j - \mathbf{q}_i\|, \|\mathbf{x}_j - \mathbf{z}_k\|)\mathbf{v}_i$, are obtained, where $\mathbf{n}_i$ and $\mathbf{v}_i$ are the normal and velocity vectors at $\mathbf{q}_i$, and the set $\{\mathbf{x}_j\}$ is the set of fragments projected onto the pixel. By normalizing these two vectors, smoothly interpolated normal and velocity vectors are obtained for each projected position on the surface.

This fact is used in the third render pass, where a single quadrilateral covering the viewport is rendered, and each generated fragment fetches the respective weighted sum of normal and velocity vectors from the `weighted_vectors` and `weighted_normals` textures. The normalized interpolated normal vector is used to compute the lit surface, which is written to the texture `lit_surface`. The normalized interpolated velocity vector is written to the texture `vectors`. The lit surface can be then displayed, or these two textures, together with the texture `intersections` can be input to the process described in the next section, where LIC is added to the lit surface.

5.3.3 LIC on the point-based surface

The point-based rendering process of the previous section provides a projection of the stream or path surface onto the image plane, along with information about

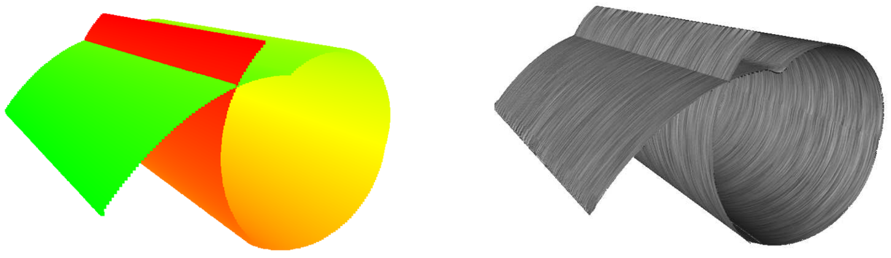


Figure 5.5: Path surface of an unsteady flow: on the left side, the time of the unsteady flow field is shown by colors (red for for early times, green for later times); on the right side the combination of the path surface and time-dependent LIC is illustrated (see color plates).

the 3D position on the surface and the attached normal and velocity vectors. The LIC computation implemented using these results can be considered a G-buffer algorithm [134] because it relies on image-space information to perform particle tracing and convolution. It is important to distinguish between the dimensionality of the domain and the dimensionality of the attached data. An image-space or G-buffer method always uses a 2D domain (the image plane), but the attached data (*i.e.*, the G-buffer attributes) can be of other dimensionality.

The hybrid object/image space method [166] needs the following G-buffer attributes: (1) the 3D position of the respective pixel in object space (in texture `intersections`) and (2) the 3D vector field in object space (in texture `vectors`). The only other data that is used for LIC is an input noise. This noise is modeled as a 3D solid texture to ensure temporal coherence under camera motion. According to Weiskopf *et al.* [166], the LIC texture I is computed by

$$I(x_I^0, y_I^0) = \int k(\tau - \tau_0) M(\mathbf{r}_O(\tau - \tau_0; x_I^0, y_I^0)) \, \mathrm{d}\tau, \tag{5.10}$$

where the subscript $_I$ denotes parameters given in image space, the subscript $_O$ denotes parameters given in object space, τ is integration time, M is the 3D noise, k is the filter kernel, and $\mathbf{r}_O$ represents positions along a pathline. The pathline is determined by the initial image-space position (x_I^0, y_I^0), which has a corresponding initial 3D object-space position on the surface at initial time τ_0.

The original implementation [166] was designed for older Shader Model 2.0 GPUs and uses multiple render passes to step along particle paths and to discretize the LIC integral. Current GPUs with Shader Model 3.0 support allow for a single-pass LIC implementation using loops. Input to this implementation are the two G-buffer textures `intersections` (object-space positions) and `vectors`

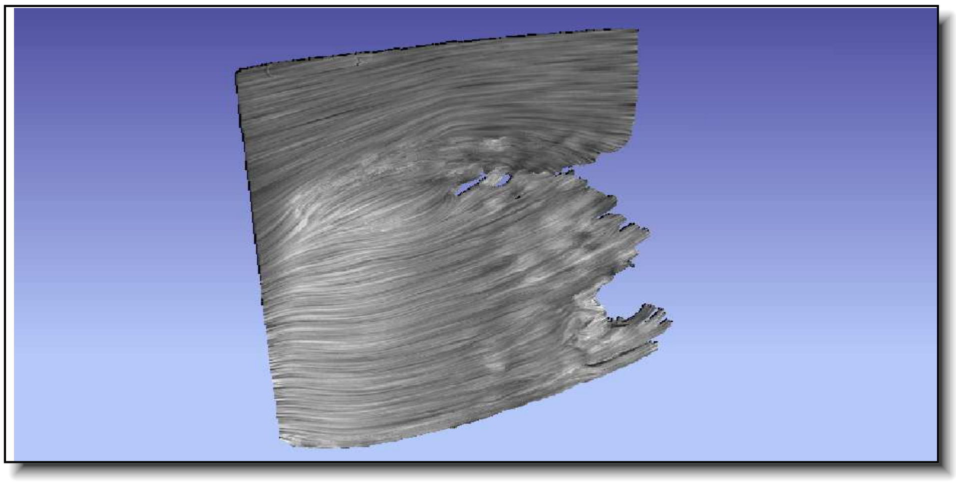


Figure 5.6: Path surface of the unsteady flow around a cylinder.

(object-space vector field), which are initialized by the point-based rendering process described above. The actual particle tracing is done in 3D object space coordinates in order to achieve higher accuracy than pure image-space advection methods. Before the vector field can be accessed, the current 3D object space position is transformed to image space by applying the model, view, and projection matrices.

LIC improves the visualization on stream or path surfaces because a LIC texture provides additional information that cannot be encoded by a surface alone. Figure 5.3 shows an example: the stream surface is quite wide and without LIC lines (Figure 5.3) the flow structure within the stream surface is not displayed; in contrast, Figure 5.3 shows the stream surface with LIC, conveying the internal flow structure and the flow direction. Here, the LIC texture is combined with the regularly rendered and illuminated surface (from `lit_surface` texture) in order to show the flow and the surface shape at the same time.

The above LIC algorithm works for steady and unsteady flow alike. Since the steady case is rather simple, the focus of the following discussion is on the unsteady scenario, which generally is challenging for texture-based flow visualization. Typically, the texture-based visualization of unsteady flow leads to smeared-out texture patterns. For example, texture advection [77; 158] constructs an overlay of streaklines (or inverse streaklines). Since streaklines may intersect each other, the weighted average of noise input from those streaklines could result in a convolution that is not restricted to a curved line. Therefore, texture patterns could be smeared out in a 2D area. Similarly, the feed forward and value depositing mechanisms of UFLIC [142; 106; 102] can lead to changing widths of line patterns.

The fundamental problem is that there is not a single, unique vector for a single spatial position in a time-dependent flow. In fact, the vector depends on how far in time the integration along a particle trace has progressed. The above texture-based methods mix, at the same position, vector fields of different time. In contrast, the surface LIC implemented obtains the vector field from path surface construction, which usually yields a single vector for a certain spatial position because that spatial position is linked to a specific time. Figure 5.5 shows a color coding of this time. Still, a path surface could intersect itself, which corresponds to two different times and two different vector values at an intersection point. Figure 5.5 illustrates such a self intersection. Fortunately, those intersection points typically form only a zero set (*i.e.*, a 1D line on a 2D surface) and lead to different flow regions in image space that are clearly separated by the intersection lines. As illustrated in Figure 5.5, surface LIC is capable of generating crisp, line-like LIC textures for those different flow regions. The proposed method was tested on a PC with a 2.21 GHz CPU and 2 GB of RAM. Two different GPUs were used: an NVIDIA GeForce 8800 GTX and an NVIDIA GeForce 7900 GTX. For the performance test, an unsteady data set was used, simulating the flow around a cylinder with 17 time steps. Figure 5.6 shows a visualization of the test data set. For the steady measurements, only the first time step is used. The vector field is given on a uniform grid of size $256 \times 128 \times 256$. Table 5.2 shows the results of a measurement with 256 particles that are integrated along 256 time steps. In the unsteady case, the vector field is updated each $256/17 \approx 15$ time steps. The performance for rendering the plain surface mainly depends on the size of the projected surface, which can be explained by the fragment-based surface approximation. Applying the surface LIC reduces the rendering speed by a factor of 2.5. Since the current driver of the new NVIDIA GeForce 8800 does not provide the full power of the architecture, particulary when rendering to texture is applied, the GPU was replaced with its predecessor, the NVIDIA GeForce 7900 GTX for measurements that need render-to-texture functionality. With that GPU, frame rates of 6.5 and 5.5 fps were measured for integrating particles in a steady and unsteady flow, respectively. Further experiments showed that the performance of particle tracing strongly depends on the size of the vector field and the number of time steps, which corresponds to the number of texture uploads.

The vector field, which is already accessed for particle integration, is stored and later reused by an image/object space LIC method to compute flow textures on the stream surfaces and path surfaces. Only this additional surface texture gives a detailed impression of the flow behavior within the surface, facilitating the identification of flow divergence and vortices, and supporting the perception of the surface shape. Stream and path surfaces have the advantage that they, by construction, “carve” a tangential vector field out of the underlying 3D flow. Therefore, a projection of the vector field and corresponding interpretation problems are

Computation	Steady	Unsteady
Surface only[1]	63.9	*
Surface with LIC[1]	26.4	*
Integration only[2]	6.5	5.5

[1] Measured with an NVIDIA 8800 GTX GPU.

[2] Measured with an NVIDIA GeForce 7900 GTX GPU.

* The rendering speed does not differ from the steady case.

Table 5.2: Performance using a $256\times128\times256$ unsteady data set with 17 time steps (in fps). Rendering speed does not depend on the underlying flow field and can be considered similar for both steady and unsteady flow. While the first two entries of the table consider only the rendering of the surface, the performance of the particle integration is given separately.

avoided. Finally, the novel combined path surface/LIC approach provides clearly defined, texture-based path-lines which is not possible with previous methods.

CHAPTER

6 MESHLESS VOLUME VISUALIZATION

Rendering volumetric data stored in structured and unstructured meshes has been addressed in the past with methods specific to each mesh type. The main difficulty in developing a unified approach is the data filtering, *i.e.*, the reconstruction of the function from the sampled data. Filtering for Cartesian grids has been widely studied [117; 153] and the interactive rendering of volumetric data stored in such grids is nowadays well documented [45]. Although filtering for other mesh types has not received the same attention within the visualization community, various rendering methods for curvilinear grids [85], tetrahedral meshes [85], adaptive-mesh-refinement meshes [161] and multiblock meshes [95] have been proposed, usually using some parameterization inside the cell in order to perform linear interpolation. However, some of this mesh types pose problems that have not been completely solved yet. For instance, adaptive-mesh-refinement meshes, where the volumetric value is stored at the center of cells of different sizes, and multiblock meshes, with multiple overlapping meshes, are cases where the interpolation of the data is not trivial (see Figure 6.1). Discontinuities and artifacts are often generated during isosurface extraction from non-conformal multiblock meshes, which are usually treated during post-processing using geometric strategies [33; 91; 140; 146; 169].

Furthermore, meshes containing mixed cell types can be found [30]. Interpolating the data in these meshes can be performed using *mean value coordinates*, proposed recently by Floater *et al.* [52], which are a generalization of barycentric coordinates (see Chapter 2). This method can effectively parametrize the domain within a cell, however only C^0-continuity is ensured across the cell boundaries and requires triangulating the faces of the polyhedron. It is possible to triangulate the faces of any polyhedron, the uniqueness of the triangulation cannot be ensured and therefore the resulting mean value coordinates depend on the choice of the triangulation.

Thus, rendering volume data using higher-order reconstructions of data stored in meshes with cells of arbitrary type is still an open problem. This problem is the focus of this chapter. A straightforward approach to solve this problem would be to disregard the mesh and treat the case as a scattered data interpolation problem. However, the connectivity information of a mesh is important and cannot be simply replaced by the spatial queries (k-nearest-neighbors, natural-neighbors and range-queries) usually performed by scattered data approximation approaches.

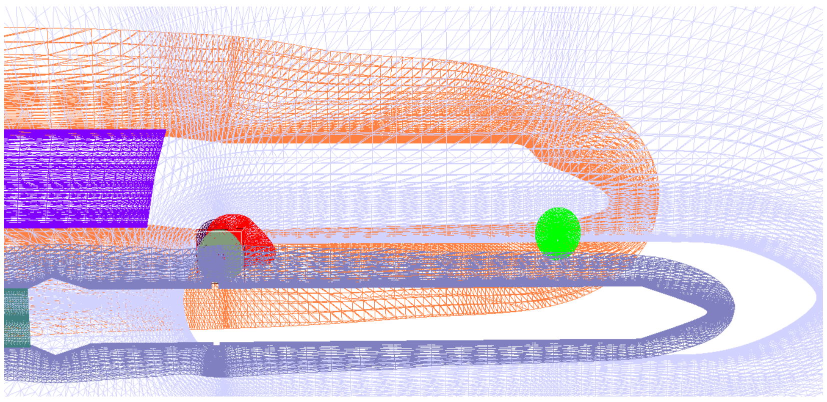


Figure 6.1: Wireframe rendering of the Space Shuttle Launch Vehicle multiblock dataset. Interpolating/approximating volumetric data stored in adaptive-mesh-refinement meshes or multiblock datasets is not trivial since the data is stored at the center of non-conformal cells of different sizes and even overlapping meshes are found.

Therefore, and since points (vertices) are too weak to represent complex data by themselves, the power of a meshless technique must be provided by the approximation method. Thus, results of approximation theory on moving least-squares, bilateral filtering, orthogonal polynomials, radial basis functions and approximate approximation are used here to approach this problem.

One important advantage of the approaches described in this chapter is that they are completely mesh-free in the sense that no mesh must be constructed as done by previous approaches, such as those based on radial basis functions, wavelets and B-splines. However, it is shown how the mesh connectivity can be used to apply the methods to highly-anisotropic domains effectively. Moreover, the methods presented are matrix-free, *i.e.*, no system of equations must be solved, which facilitates its implementation on commodity graphics hardware. This work was developed in collaboration with João Paulo Gois from the Universidade de São Paulo.

The problem of meshless volume deformation is also addressed in this chapter and the advantages of moving least-squares in the context of volume manipulation to support exploratory tasks are studied. To that end, recent results on non-physically-based moving least-squares deformation are used and an interactive hardware-accelerated implementation applicable to structured and unstructured grids is presented. A comparison with physically-based tetrahedral mesh deformation is presented in terms of interactivity, for which graphics hardware implementations were developed for both the moving least-squares and the physically-based deformations. The work on moving least-squares volume deformation was developed in collaboration with Alvaro Cuno from the Universidade Federal do

Rio de Janeiro and Siegfried Hodri from the Universität Stuttgart.

6.1 Meshless Methods for Volume Visualization

Jang *et al.* [76] and Weiler *et al.* [163] used radial basis functions to encode scalar and vector fields stored in structured and unstructured meshes. Since a functional representation is obtained, evaluating derived measures is straightforward. The authors use truncated Gaussian functions as basis functions. To accurately represent local features, the widths of each truncated Gaussian is adaptively specified. Thus, the functional representation is given by

$$f(\mathbf{x}) = \omega_0 + \sum_{i=1}^{M} \omega_i \exp\left(\frac{\|\mathbf{x} - \mu_i\|^2}{2\sigma_i^2}\right),$$

where M is the number of radial basis functions, ω_0 is the bias and σ_i^2, ω_i and μ_i are the width, the weight and the center of the radial basis function respectively. To determine the center location, the authors make use of *principal components analysis* to cluster the data points and, in each cluster, the center is selected as the weighted cluster average point or the maximum error point as chosen by the user. The width is determined by a hybrid gradient-descent nonlinear optimization technique (Levenberg-Marquardt method). The mean square error over all data points is used during optimization. The individual weight and global bias are computed by minimizing the sum squared error for all data points. Using this method, radial basis functions are incrementally added in clusters with the largest errors until the user specified error criteria is satisfied. Encoding errors are calculated as the difference between the original value and the evaluated radial basis function representation at each input data point.

Jang *et al.* [75] extended their work to ellipsodial basis functions in order to reduce the number of basis functions required to encode a volume and to better reconstruct data where long features are present. They explore the use of axis-aligned and arbitrary directional ellipsoidal basis functions. Using the *Mahalanobis distance*, the ellipsoidal basis function, specifically the ellipsoidal Gaussian function, in three dimensions can be represented in matrix form as

$$\omega_{EBF}(\mathbf{x}, \mu, \mathbf{V}^{-1}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^T \mathbf{V}^{-1}(\mathbf{x} - \mu)\right),$$

where μ is the center and $\mathbf{V}^{-1}$ is positive definite and is defined by a rotation matrix $\mathbf{R}$ and a scaling matrix $\mathbf{S}$ as $\mathbf{V}^{-1} = \mathbf{R}\dot{\mathbf{S}}^{-1}\dot{\mathbf{S}}^{-1}\dot{\mathbf{R}}^T$. The parameters of $\mathbf{V}^{-1}$ are found performing a nonlinear optimization of the sum of the squared error as before.

Lee *et al.* [96] approach the problem using B-splines. Although good results are obtained, the need for a regular grid of control points is a major drawback.

This is also the case for Wavelet-based methods [61]. More flexible splines defined on more general domains have been also proposed but numerical problems arise and computational costs are significantly increased. Moreover, underlying connectivity information is still necessary. Rössl *et al.* [132] also presented a scattered data interpolation method based on splines and local least-squares built upon Bernstein-Bézier basis and a tetrahedral spatial decomposition. One of the main computational efforts of the method is related to the singular value decomposition needed for each least-squares computation, which is repeated until an error criterion is satisfied.

Anderson *et al.* [12] approached the problem of visualizing volumetric data stored in tetrahedral meshes by defining ellipsoidal point primitives. Differently from the work by Jang *et al.*, the focus of the method presented by Anderson *et al.* is on rendering and not on function reconstruction. The algorithm is comprised of three steps. The first step is a pre-processing where the points are created; each point having a representative transform and scalar value associated with it. An entire tetrahedron is thus represented by a single point. The scalar value at the point is defined as the mean value of the scalar data stored at the vertices of the tetrahedron. Since the point primitives are rasterized as squares, their shape must be transformed to better approximate the tetrahedra they represent. The transform is obtained by defining a regular tetrahedron centered at the origin such that it is inscribed in the unit sphere. The transform is then calculated as the transformation T that transforms the regular tetrahedron to the tetrahedron being processed. In the second step, the points are sorted front-to-back on the CPU. Then, in the third step, the point primitives are rendered and the vertex program resizes each point using its transform to ensure that the tetrahedron is adequately represented. The fragment program culls fragments based on the shape of the approximating element. Pre-integration is then used to better approximate the volume rendering integral.

Points had been previously used to render volumetric data [135]. The authors proposed a method that renders a set of tiny particles generated taking into account a user-specified transfer function. The main advantage of this technique, based on the emission-absorption model, is that the particles can be projected onto the image plane without performing any sorting process. As seen in Chapter 2, the volume rendering integral for the emission-absorption model has the form

$$L(D) = L_0 \exp\left(-\int_0^D \tau(t)dt\right) + \int_0^D L_e(s)\tau(s)\exp\left(-\int_0^s \tau(t)dt\right)ds.$$

If $L_0 = 0$ then,

$$L(D) = \int_0^D L_e(s)\tau(s)\exp\left(-\int_0^s \tau(t)dt\right)ds$$

is obtained. This integral can be solved numerically by subdividing the domain of integration into n sub-domains in which the emission L_e can be regarded as constant, which gives us

$$L(D) = L_1 + L_2 + L_3 + \cdots + L_k + \cdots + L_n,$$

where

$$L_k = \int_{t_{k-1}}^{t_k} L_e^{[k]} \tau(s) \exp\left(- \int_0^s \tau(t)dt \right) ds$$

and $L_e^{[k]}$ is the constant emission in the k-th sub-domain. An opacity value α_k, in the k-th subdomain, can be defined as

$$\alpha_k = 1 - \exp\left(- \int_{t_k}^{t_{k+1}} \tau(t)dt \right) \approx 1 - \exp(-\tau_k \Delta t),$$

where Δt is the length of the subdomain and τ_k is a representative density. Usually, the opacity is specified from scalar values. A particle density relates to a scalar value $f(\mathbf{x})$ implicitly as

$$\begin{aligned} \alpha(f(\mathbf{x})) &= 1 - \exp\left(-\tau(\mathbf{x})\Delta t\right), \\ \tau(\mathbf{x}) &= \begin{cases} \frac{-\log(1-\alpha(f(\mathbf{x})))}{\Delta t} & \alpha(f(\mathbf{x})) \leq 1 - \exp(-\Delta t) \\ 1.0 & \text{otherwise}, \end{cases} \end{aligned}$$

where $\mathbf{x}$ is a position on the viewing ray. Thus, the particle density can be determined from an opacity value converted from a scalar value using the transfer function. The particles are generated according to the density distribution function $\tau(\mathbf{x})$ given above using the *hit-and-miss* or *metropolis* methods. The image is then created by projecting the generated particles onto the image plane. On the image plane, each pixel can be divided into several sub-pixels. The color of each pixel is determined by averaging the colors of its sub-pixels.

6.2 Moving Least-Squares Volume Visualization

In this section, a method based on moving least-squares to render volumetric data using higher-order approximations, that can be applied to meshes of arbitrary geometry and topology, is presented. To preserve important details in the data, the approximation method by Fenn *et al.* [50], based on bilateral filtering, is extended to three dimensions. Although extending this method to the three-dimensional domain is in theory straightforward, this leads to an ill-conditioned problem for which basic numerical methods diverge. Pre-conditioning and regularization can be used together with expensive more stable methods to solve this problem. However, it was approached using multi-variate orthogonal polynomials since this allows to avoid solving systems of equations while improving stability. Furthermore, performance is increased since orthogonal polynomials present properties that can be used to reduce the number of operations needed to calculate the approximation. Also, as in the case of surface approximation (Chapter 4), the recursive nature of the revised Gram-Schmidt orthogonalization used can be exploited to increase the degree of the approximating polynomial using the results of previously computed lower-degree polynomials. Thus, the degree of the polynomial can be set adaptively without incurring in a high extra computational effort as would be the case with other methods.

An empirical comparison, in the context of the method proposed, in terms of stability and performance between orthogonal polynomials and methods applied both on the overdetermined system and on the normal equation was carried out. Specifically, experiments with the Conjugate Gradient (CG) method on normal equations, Singular Value Decomposition (SVD), QR factorization via Householder transformations and Gauss-Jordan (GJ) with pivoting applied on normal equations [20] were performed. As will be shown, orthogonal polynomials outperform these methods in terms of speed of computation while being sufficiently stable for the purposes of the problem at hand. Additionally, to improve the quality of the approximation obtained with the method presented, ellipsoidal weights are used in the moving least-squares formulation presented. Results using Cartesian grids, unstructured meshes, curvilinear grids, adaptive-mesh-refinement and multiblock datasets with multiple overlapping meshes are presented.

6.2.1 Detail-preserving volume data approximation

The problem addressed here is the general approximation problem described in Chapter 3 in the context of volumetric data. Thus, here the focus is on the three-dimensional case. Given a set of sample points $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\} \subsetneq \mathbb{R}^3$, and a function $f : \mathbb{R}^3 \to \mathbb{R}$ evaluated on $\mathcal{X}$, generating the set $\mathcal{F} = \{f_1, \ldots, f_N\} \subsetneq \mathbb{R}$, find a function $\mathcal{M}f$ such that $\mathcal{M}f(\mathbf{x}_j) \approx f_j$; $j = 1, \cdots, N$.

As seen in Chapter 3, given a polynomial basis $\Psi = \{\psi_1, \ldots, \psi_M\}$, a moving least-squares polynomial approximation

$$\mathcal{M}f(\mathbf{x}) = \sum_{j=1}^{M} c_j(\mathbf{x})\psi_j(\mathbf{x})$$

to a function f is found by minimizing

$$\min E(\mathbf{x}) = \sum_{i=1}^{N} \left(f_i - \mathcal{M}f(\mathbf{x}_i)\right)^2 \omega(\mathbf{x}, \mathbf{x}_i), \tag{6.1}$$

with respect to c_j; $j = 1, \cdots, N$. When computing a detail-preserving local approximation, on the other hand, the aim is to minimize the following energy functional:

$$\min R(\mathbf{x}) = \sum_{i=1}^{N} \Upsilon\left((f_i - \mathcal{M}f_{RMLS}(\mathbf{x}_i))^2\right)\omega(\mathbf{x},\mathbf{x}_i), \tag{6.2}$$

where

$$\Upsilon(p) = 1 - \exp\,(-\frac{p}{2\sigma_\rho})$$

and σ_ρ is a parameter that determines the sensitivity of the expression to outliers. Other functions Υ can be found in the literature [50; 155]. The goal of this minimization is to consider not only the distance between the evaluation point $\mathbf{x}$ and the sample points $\mathbf{x}_i$; $i = 1, \cdots, N$, but also the difference between f_i and $\mathcal{M}f_{RMLS}(\mathbf{x}_i)$ for weighting the influence of the sample points on the result. In this case the system of equations that minimizes $R(\mathbf{x})$ becomes

$$\left\{\sum_{j=1}^{M} \langle \Psi_k, \Psi_j\rangle_{\omega\Upsilon'} c_j = \langle \Gamma, \Psi_k\rangle_{\omega\Upsilon'};\ k = 1, \ldots, M, \right. \tag{6.3}$$

where $\Gamma = [f_1, \cdots, f_N]$, $\Psi_j = [\psi_j(\mathbf{x}_1), \ldots, \psi_j(\mathbf{x}_N)]$, Υ' is the first derivative of Υ and

$$\langle \Psi_j, \Psi_k\rangle_{\omega\Upsilon'} = \sum_{i=1}^{N} \psi_j(\mathbf{x}_i)\psi_k(\mathbf{x}_i)\omega(\mathbf{x},\mathbf{x}_i)\Upsilon'((\mathcal{M}f_{RMLS}(\mathbf{x}_i) - f_i)^2).$$

The solution of System 6.3 can be obtained with a fixed point iteration as follows. Let

$$\mathcal{M}f^{(n)}_{RMLS}(\mathbf{x}) = \sum_{j=1}^{M} c_j^{(n)}(\mathbf{x})\psi_j(\mathbf{x})$$

be the solution for iteration n, and $(c_1^{(0)}(\mathbf{x}), \ldots, c_M^{(0)}(\mathbf{x}))$ an initial guess obtained by solving System 3.4, rewritten here for convenience:

$$\left\{\sum_{j=1}^{M} \langle \Psi_i, \Psi_j\rangle_{\omega} c_j = \langle \Gamma, \Psi_i\rangle_{\omega};\ i = 1, \ldots, M \right. \tag{6.4}$$

Therefore, the inner products are fixed, for iteration n, as

$$\langle \Psi_j, \Psi_k\rangle^{(n)}_{\omega\Upsilon'} = \sum_{i=1}^{N} \psi_j(\mathbf{x}_i)\psi_k(\mathbf{x}_i)\omega(\mathbf{x},\mathbf{x}_i)\Upsilon'((f_i - \mathcal{M}f^{(n-1)}_{RMLS}(\mathbf{x}_i))^2),$$

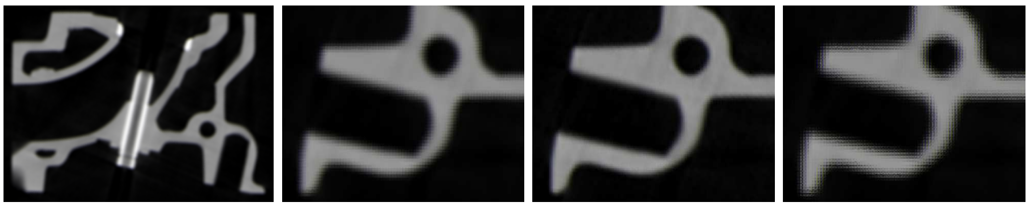


Figure 6.2: Slice 82 of the Engine volume dataset (left). Whilst moving least-squares smooths the data (center-left), the robust approximation preserves the details (center-right). Shepard's interpolation (right) is shown for sake of comparison.

and, thus, the solution for iteration n is obtained by solving the system

$$\left\{ \sum_{j=1}^{M} \langle \Psi_k, \Psi_j \rangle_{\omega\Upsilon'}^{(n)} c_j^{(n)} = \langle \Gamma, \Psi_k \rangle_{\omega\Upsilon'}^{(n)};\ k = 1, \ldots, M \ . \right. \tag{6.5}$$

Then, for each iteration, a moving least-squares approximation is performed to find a new $(c_1^{(n+1)}(\mathbf{p}), \ldots, c_M^{(n+1)}(\mathbf{p}))$. Although convergence is not proven for this iterative process, in practice 5 iterations suffice to obtain a good approximation if the initial guess is computed by solving System 6.4. Note that in the above discussion the sums are on the entire set $\mathcal{X}$. However, as before, in practice only a subset of $\mathcal{X}$ is used to approximate the function at point $\mathbf{x}$. This subset is defined as the neighborhood of $\mathbf{x}$, that usually is given by the k nearest neighbors, where $k > M$.

The implementation of this robust approximation is straightforward. The first approximating function computed by solving System 3.4 is used as starting point for the iterative process given by System 6.5.

6.2.2 Matrix-free detail-preserving volume data approximation

Although the process described in the previous section results in a detail-preserving approximation of the volumetric data (see Figure 6.2), a very large number of small systems of equations must be solved. More importantly, the systems of equations are ill-conditioned and solving them with known techniques such as Gauss-Jordan (with pivoting) or Conjugate Gradient leads to instabilities. More expensive techniques, such as preconditioned Conjugate Gradient or SVD and QR factorizations with regularization can be used to improve stability. Good results are obtained with these methods as will be discussed. However, the computational effort is prohibitive and increasing the degree of the approximation means having to compute the new approximation and to discard the previously computed approximation. Orthogonal polynomials, on the other hand, are well suited for this task due to their recursive construction while being competitive in terms of stabil-

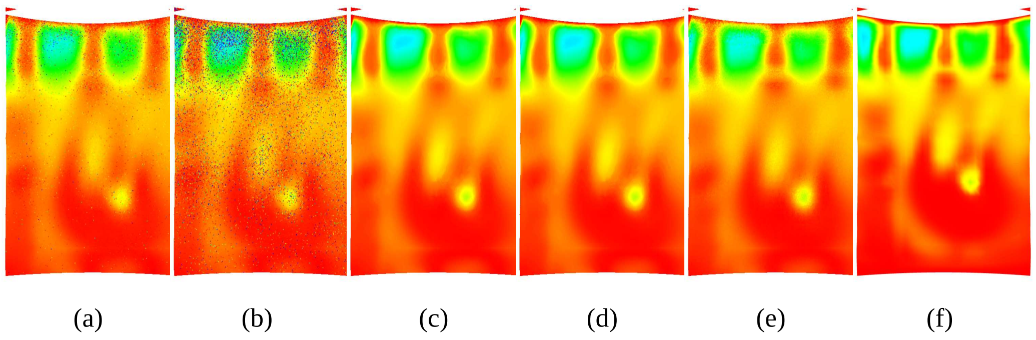


Figure 6.3: Volumetric data approximation for a slice of the Combustion Chamber dataset: (a) Gauss-Jordan with pivoting, (b) Conjugate Gradient on normal equations, (c) QR, (d) SVD and (e) orthogonal polynomials. Noise represents evaluation points where instabilities led to a poor approximation. In (f), the result with linear interpolation on the original mesh is shown. Note that, for these results, only `float` precision was used to increase the numerical instability. In practice, choosing a sufficiently large support and using `double` precision, orthogonal polynomials produce results visually indistinguishable from those obtained with QR and SVD (see color plates).

ity compared to expensive techniques and in terms of performance compared to basic techniques.

Note that, in order to be able to define orthogonal polynomials, Υ and ω must hold $\Upsilon' > 0$ and $\omega > 0$ ensuring that $\langle , \rangle^{(n)}_{\omega\Upsilon'}$ defines a new inner product for each iteration. Using orthogonal polynomials, the robust approximation is also an iterative process. The approximation in the first iteration is obtained using the inner product $\langle , \rangle_\omega$, where the weight function ω is positive and decreasing. Based on this weight a first set of orthogonal polynomials is found using the process described in Chapter 3. The first approximation to the solution is then given by

$$\mathcal{M}f^{(0)}_{RMLS}(\mathbf{x}) = \sum_{i=1}^{M} \psi_i(\mathbf{x}) \frac{\langle \Gamma, \Psi_i \rangle_\omega}{\langle \Psi_i, \Psi_i \rangle_\omega}.$$

With this first result the iterative process is started. In each iteration n a new set of orthogonal polynomials is constructed using the inner product $\langle , \rangle^{(n)}_{\omega\Upsilon'}$ and a new approximated solution is found as

$$\mathcal{M}f^{(n)}_{RMLS}(\mathbf{x}) = \sum_{i=1}^{M} \psi_i(\mathbf{x}) \frac{\langle \Gamma, \Psi_i \rangle^{(n)}_{\omega\Upsilon'}}{\langle \Psi_i, \Psi_i \rangle^{(n)}_{\omega\Upsilon'}}.$$

The implementation of the method was tested with regular volumes (Cartesian grids), structured curvilinear grids, tetrahedral meshes, adaptive-mesh-refinement meshes and multiblock datasets. From the tests performed it was found that the

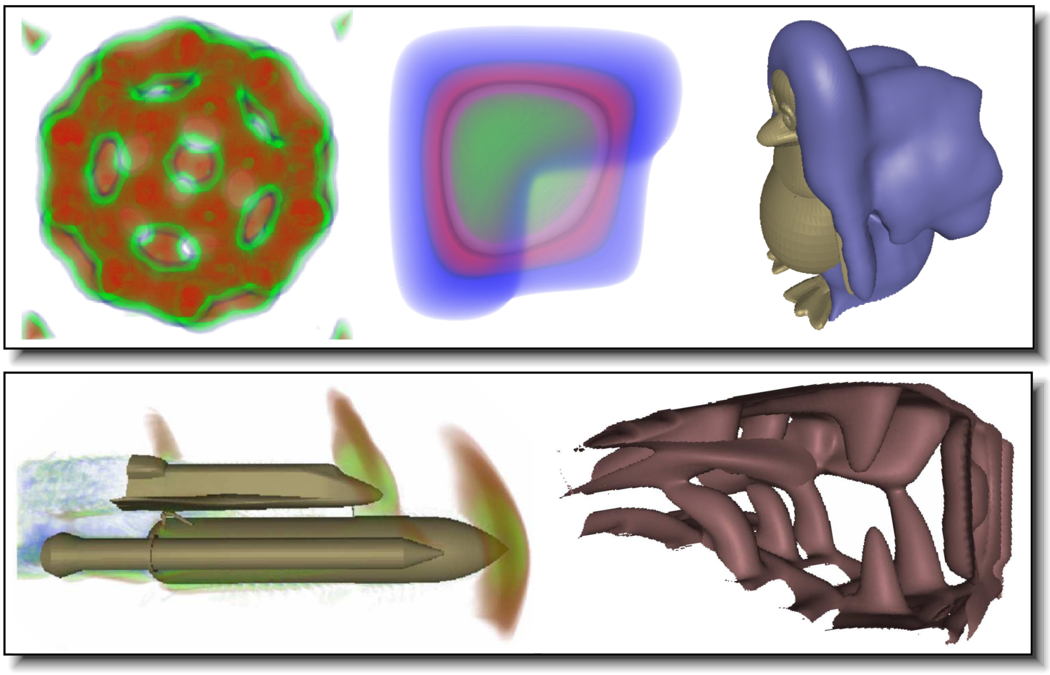


Figure 6.4: Volume and isosurface rendering of different data. From left to right: the Bucky Ball structured dataset, the Heat Sink unstructured dataset, the Penguin adaptive-mesh-refinement mesh, the Space Shuttle Launch Vehicle multiblock dataset with multiple overlapping curvilinear meshes and the Combustion Chamber curvilinear dataset (see color plates).

approximation reconstructs the function preserving details while filtering low frequency noise. This can be seen in Figure 6.2, where a comparison of Shepard's interpolation, moving least-squares approximation and the detail-preserving approximation is shown. Although Shepard's interpolation is faster, it fails to reconstruct the function accurately, while moving least-squares smoothes the data.

As mentioned before, several solvers were implemented besides the approach based on orthogonal polynomials. Specifically, SVD and QR factorization methods were used due to their stability, as well as the Conjugate Gradient on normal equations (see Section 13 of the report by Schewchuk [145]) and Gauss-Jordan with pivoting due to their simplicity and high performance. Visual results of the approximation obtained with these methods for a slice of the Combustion Chamber dataset are shown in Figure 6.3. For sake of comparison, the resulting linear interpolation on the original mesh obtained with barycentric coordinates is also shown. For this test, all `double` variables where changed to `float` to increase the probability of incurring in numerical instabilities. It is important to mention that, although a cutting plane is shown, the approximation was performed in the three dimensional domain. As can be seen in the figure, the SVD and QR fac-

	Method				
Polynomial degree	Orth. Pol	CG	Gauss-Jordan	QR	SVD
2	0.19	0.43	0.76	0.46	0.87
3	0.47	0.61	0.93	1.15	2.84
4	0.76	0.82	1.83	2.19	8.35

Table 6.1: Processing time (in milliseconds), for a single evaluation of the approximation, with orthogonal polynomials, Conjugate Gradient (CG), Gauss-Jordan, and QR and SVD factorizations.

torizations are the most stable followed by orthogonal polynomials. Conjugate Gradient and Gauss-Jordan with pivoting are not able to perform well due to the high condition number of the matrix. It is important to remark that Gauss-Jordan and orthogonal polynomials are used to solve Systems 6.4 and 6.3 while SVD, CG and QR are directly applied on the overdetermined systems that result from Equations 6.1 and 6.2 [20].

Although the SVD and the QR factorizations are more stable than orthogonal polynomials, they are considerably slower. Table 6.1 summarizes the performance measurements carried out. Since the function approximation does not depend on the mesh type, the performance for all datasets was similar. The performance measurements were carried out on a standard PC equipped with a 3.4GHz 64-bit processor and 2GB of RAM. As can be seen, orthogonal polynomials are faster than any other method tested. Furthermore, orthogonal polynomials have a clear advantage when the degree of the polynomial approximation increases. This can also be observed in the table, which shows the computation time for approximations of degrees 2 to 4. This is possible due to the recursive nature of the revised Gram-Schmidt orthogonalization process, which allows to use previously computed low-degree polynomial approximations to obtain a higher degree polynomial. This means that the processing time in Table 6.1 for the orthogonal polynomials of degree 3, for instance, includes the computation of the orthogonal basis for polynomial approximations of degree 0 to 3.

To accommodate the method to anisotropic meshes, ellipsoidal weight functions are used. This allows to handle meshes with irregular sampling for which spherical weights do not present good results as can be seen in Figure 6.5. It is important to notice that, although an ellipsoidal Gaussian weight is used, in practice, its support is truncated to the ellipsoidal region defined by the neighborhood of the evaluation point. Using double precision and the ellipsoidal weight functions, the orthogonal polynomials proved to be stable, accurate and fast to compute. As a proof of concept, a ray-caster was implemented to generate direct volume renderings and isosurface renderings of the test datasets. The results can be seen in

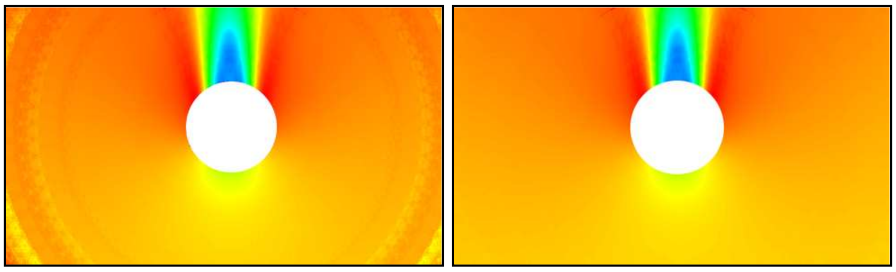


Figure 6.5: Comparison between the use of spherical (left) and ellipsoidal (right) weights (Oxygen Post dataset).

Figure 6.4 where renderings of different datasets are shown. Note that the isosurfaces are smooth even for the adaptive-mesh-refinement and multiblock datasets with no need for post-processing or special handling.

Orthogonal polynomials have shown to be a good choice due to their stability in volumetric data approximation, while being faster than any of the other four approaches implemented to solve the systems of equations. Several advantages of using orthogonal polynomials were found, namely, the fact that the method becomes matrix-free and the recursive nature of the Gram-Schmidt orthogonalization. This allows us to reconstruct the underlying function of the data stored in meshes of any type preserving details in the data by means of bilateral filtering and ellipsoidal weight functions. Also, if the degree of the polynomial approximation is increased, the new polynomials that must be added to the set of orthogonal polynomials can be calculated using previously computed polynomials. Thus, the degree of the polynomial could be increased by the user until the desired approximation order is achieved.

Although the fact that the method is matrix-free, *i.e.*, that no system of equations must be solved, was exploited to implement it on commodity graphics hardware, it was found that, despite being faster than solving the system, calculating the orthogonal polynomials is still slow for visualization purposes. Therefore, approximate approximations were explored to address this problem as will be seen in the next section.

6.3 Approximate MLS Volume Visualization

In this section, a further volume rendering method that can be applied to meshes of arbitrary geometry and topology is presented. This method is based on the iterated approximate moving least-squares approximation [48] which is applied to anisotropic domains to reconstruct the volumetric data during rendering in this work. This allows to preserve important details on the data, in the same spirit as in surface reconstruction from point clouds [41]. To provide gradient information for shading and other visualization purposes, the gradient at each vertex of the mesh is estimated by means of weighted least-squares [110]. During rendering, the gradient at the evaluation point is reconstructed using iterated approximate approximation as done for the raw volumetric data. It is worth to mention that iterated approximate moving least-squares approximation with anisotropic weights has not been previously addressed. In this work, it is shown how this combination is possible and how efficient it is in the context of volume rendering.

Although a meshless interpolation method is used, anisotropic meshes containing highly stretched cells are handled properly and automatically by means of ellipsoidal weight functions calculated from mesh information. For this, the construction of the ellipsoidal supports are explored in more detail here. The local ellipsoidal weight functions make the method robust and suitable for different meshes. The results show that no parameter tuning is necessary since the same values produced good results. The implications of using ellipsoidal weights in the iterated approximate moving least-squares approximation are also discussed and an adaptive iterative approximate approximation is defined.

In order to accelerate the method, a GPU version was implemented based on spatial data structures which is able to handle non-convex meshes with highly stretched cells. It is shown how to pre-compute the result of the iterative process at the sample points, so that during rendering the method reduces to a weighted sum of the data stored at the vertices.

Additionally, the fact that no system of equations must be solved can be mentioned as an advantage of the method. Also, as discussed in the previous section, no pre-processing to define a new topological data structure of the data is needed, such as done in methods based on radial basis functions, wavelets and B-splines.

6.3.1 Ellipsoidal weight functions revisited

As mentioned in the last section, ellipsoidal weight functions produce better reconstruction results for moving least-squares approximations of volumetric data. In this section, it is further discussed how ellipsoidal supports can be used in the context of approximate approximation. Since Gaussian weight functions are used, in practice, the weight function is truncated to force compact support. Special care must be taken into defining this support so as to ensure that any significant contribution from the sample points to the reconstructed value is always taken into account. This is achieved by choosing a suitable fill size parameter. With this consideration, the ellipsoidal support is computed with a straightforward approach.

Given a sample point $\mathbf{x}_i$, the set $\mathcal{V}_i = \{\mathbf{x}_j \in \mathcal{X} : \mathbf{x}_j \in \mathrm{star}(\mathbf{x}_i)\}$ is computed, *i.e.*, the set of points that share an edge in the mesh with $\mathbf{x}_i$. These points are used as input for a principal components analysis. For this, the covariance matrix

$$\mathbf{C} = \sum_{\mathbf{x} \in \mathcal{V}_i} (\mathbf{x} - \mathbf{x}_i) \otimes (\mathbf{x} - \mathbf{x}_i)$$

is calculated. The eigenvectors $\mathbf{e}_k; k = 1, \cdots, 3$ of $\mathbf{C}$ are the main directions of the ellipsoid. The eigenvalues can be used to define the support of the ellipsoidal weight. Here, however, the maximal distance in each of the main directions between $\mathbf{x} \in \mathcal{V}_i$ and $\mathbf{x}_i$ is used, *i.e.*, $\eta_k = \kappa \max_{\mathbf{x} \in \mathbb{V}_i} |\langle \mathbf{x} - \mathbf{x}_i, \mathbf{e}_k \rangle|$, where η_k defines the size of the support in the direction $\mathbf{e}_k$. Thus, the transformation matrix $\mathbf{M}$ that defines the local support is given by

$$\mathbf{M} = \operatorname{diag}(\eta_1, \eta_2, \eta_3) \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}_3 \end{bmatrix} .$$

This support is used to define the ellipsoidal Gaussian weight as explained below. The constant κ scales the support in all directions. This constant hast to be set so as to include enough data points according to the order of the generating function used. The role of this constant will be further discussed later in this section.

Note that this computation is similar to the one used by Jang *et al.* [75]. However, since they need to compute the center of the support, a non-linear system must be solved for each support, which increases the computational cost. Another important difference is the use of an stencil obtained from the mesh information. This helps to adapt the method to both highly anisotropic and isotropic meshes, as will be shown in the results. Thus, differently from Jang, it is not necessary to decide between using axis-aligned ellipsoidal weights, arbitrarily oriented ellipsoidal weights or spherical weights for each data set. The approach used to compute the weight function using the stencil from the mesh has shown to be reliable and to produce good results for all the test meshes used. The anisotropic support used is also similar to the one proposed by Dinh *et al.* [41] for reconstructing surface from point clouds. The authors also use anisotropic basis functions built upon covariance analysis. They argue that by defining suitable anisotropic functions on the data, it is possible to represent sharp details of the original object.

6.3.2 Anisotropic iterated approximate moving least-squares

The use of anisotropic spaces is not new and it has delivered very satisfactory results in the past, *e.g.*, in mesh generation [144], in scalar function encoding [75] and in surface reconstruction [41]. However, the use of anisotropic spaces in iterated approximate approximation is, to the extent of our knowledge, discussed for the first time here. It is worth to mention that, despite the lack of a rigorous mathematical proof regarding convergence, numerical results show good results. The importance of iterated approximate approximation applied on anisotropic spaces is that it can generate a good reconstruction of the volumetric data without the need for solving systems of equations and no new space decomposition model of the data must be generated. Thus, it is very simple to implement on modern commodity graphics hardware.

The original iterated approximate moving least-squares method was defined under the canonical inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^t \mathbf{y}$. By simply recalling the results presented by Fasshauer and Zhang [48], it is not difficult to verify that these results can also be extended to the inner product

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{A}} = \mathbf{x}^t \mathbf{A} \mathbf{y},$$

induced by a positive matrix $\mathbf{A}$. To apply this to the problem of rendering arbitrary meshes, the inverse $\mathbf{R}_i$ of the transformation matrix $\mathbf{M}_i$ is calculated for the support of each sample point as described above. Then, a symmetric positive matrix

$$\mathbf{A}_i = \mathbf{R}_i \mathbf{R}_i^T$$

is defined for each sample, which induces a local inner product $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{A}_i}$, since $\mathbf{A}_i$ is positive-definite. The implications of this inner product definition in the practical set of the problem addressed can be better understood by looking to a specific case. Thus, let the Laguerre-Gaussian $\varphi_i(\mathbf{x})$ be the generating function for each sample point $\mathbf{x}_i$, as in Chapter 3, so that the approximated value at $\mathbf{x}$ is given, as before, by

$$\mathcal{M} f_{VAMLS}(\mathbf{x}) = \sum_{i=1}^{N} f_i \varphi_i(\mathbf{x}).$$

In three dimensions, the generating functions φ_i are given, for order $O(h^2)$, $O(h^4)$ and $O(h^6)$, by

$$\varphi_i(\mathbf{x}) = \frac{\epsilon^3}{\pi^{3/2}} \exp\left(-r_i(\mathbf{x})\right),$$

$$\phi_i(\mathbf{x}) = \frac{1}{\pi^{3/2}} \left(\frac{5}{2} - r_i(\mathbf{x}) \right) \exp\left(-r_i(\mathbf{x})\right),$$

and

$$\phi_i(\mathbf{x}) = \frac{1}{\pi^{3/2}} \left(\frac{35}{8} - \frac{7}{2} r_i(\mathbf{x}) + \frac{1}{2} r_i(\mathbf{x})^2 \right) \exp\left(-r_i(\mathbf{x})\right),$$

respectively. The function $r_i(\mathbf{x})$ is redefined as

$$r_i(\mathbf{x}) = \frac{\epsilon^2 \hat{\mathbf{x}}_i \mathbf{A}_i \hat{\mathbf{x}}_i^T}{h^2},$$

where $\hat{\mathbf{x}}_i = \mathbf{x} - \mathbf{x}_i$. Thus, the term $\hat{\mathbf{x}}_i \mathbf{A}_i \hat{\mathbf{x}}_i^T$ can be regarded as the squared distance between $\mathbf{x}$ and $\mathbf{x}_i$ in the space with origin at $\mathbf{x}_i$ defined by the transformation matrix $\mathbf{R}_i$ (recall that this distance is also known as Mahalanobis distance). Thus, κ fulfills the role of the fill size and the actual fill size h can be set to a constant, *e.g.,* 0.25, which ensures that all data points in the support are considered for the approximation. Note that using the Mahalanobis distance, the effect obtained is that of a local anisotropic fill size. The iterated process that converges, in this case to an ellipsoidal basis functions interpolation, is given by

$$\mathcal{M}f^{(0)}_{VMLS}(\mathbf{x}) = \sum_{i=1}^{N} f_i \varphi_i(\mathbf{x}), \tag{6.6}$$

$$\mathcal{M}f^{(n+1)}_{VMLS}(\mathbf{x}) = \mathcal{M}f^{(n)}_{VMLS}(\mathbf{x}) + \sum_{i=1}^{N} \left(f_i - \mathcal{M}f^{(n)}_{VMLS}(\mathbf{x}_i)\right) \varphi_i(\mathbf{x}). \tag{6.7}$$

6.3.3 Gradient estimation

The least-squares gradient estimation by Mavriplis [110], which is independent of the topology of the mesh, is generalized here to three dimensions. The method relies on defining a stencil which identifies relevant points on the star of the vertex where the gradient must be evaluated. Let us consider a point $\mathbf{x}_i = (x_i, y_i, z_i)$ where the derivative must be evaluated and $\mathbf{x}_k = (x_k, y_k, z_k) \in \mathcal{V}(\mathbf{x}_i)$. The following weighted least-squares allows us to estimate the gradient $\nabla f(\mathbf{x}_i) = ((f_x)_i, (f_y)_i, (f_z)_i)$:

$$\min \sum_{\mathbf{x} \in \mathcal{V}_i} \exp(-\langle \mathbf{x}_k - \mathbf{x}_i, \mathbf{x}_k - \mathbf{x}_i \rangle_{A_i})(E_{ik})^2, \tag{6.8}$$

where

$$(E_{ik})^2 = ((f_x)_i \cdot dx_{ik} + (f_y)_i \cdot dy_{ik} + (f_z)_i \cdot dz_{ik} - df_{ik}), \tag{6.9}$$

$df_{ik} = f(\mathbf{x}_k) - f(\mathbf{x}_i)$ and, analogously, $dx_{ik} = x_k - x_i, dy_{ik} = y_k - y_i, dz_{ik} = z_k - z_i$. Since the number of neighbors, in general, is small and the point positions can be arbitrarily defined, singular value decomposition is used to find the unknowns $((f_x)_i, (f_y)_i, (f_z)_i)$ more precisely. Thus, the gradient at any point $\mathbf{x}$ in the domain is obtained as

$$\nabla f(\mathbf{x}) = \sum_{i=1}^{N} \nabla f(\mathbf{x}_i) \varphi_i(\mathbf{x}).$$

Similarly to the volumetric data, the gradient approximation is obtained with iterated approximation as

$$\nabla f^{(0)}_{VMLS}(\mathbf{x}) = \sum_{i=1}^{N} \nabla f(\mathbf{x}_i) \varphi_i(\mathbf{x}),$$

$$\nabla f^{(n+1)}_{VMLS}(\mathbf{x}) = \nabla f^{(n)}_{VMLS}(\mathbf{x}) + \sum_{i=1}^{N} \left(\nabla f(\mathbf{x}_i) - \nabla f^{(n)}_{VMLS}(\mathbf{x}_i)\right) \varphi_i(\mathbf{x}).$$

6.3.4 GPU-based rendering

As mentioned before, modern commodity graphics hardware is used to accelerate the rendering process. Here we describe the pre-computations performed to reduce the rendering time, as well as the data structures, their map to textures stored in graphics memory and the render passes implemented for the volume rendering engine.

Pre-processing. The data points $(\mathbf{x}_i, f_i)$ are stored in a two-dimensional texture of size $\sqrt{N}$. The gradients at the vertices of the mesh are then pre-computed as described above. This allows to perform shading and enable the use of other visualization methods based on gradient information. During rendering, the gradients are reconstructed at the evaluation point using the adaptive iterated approximate approximation as done for the raw volumetric data. This is done since the results by Fasshauer and Zang [48] do not apply to derived data. The pre-computed gradients are packed into a two-dimensional texture of the same size of the texture holding the positions and data of the vertices of the mesh. The matrix $\mathbf{A}_i$ is also pre-computed for each vertex of the mesh, which is then stored in three further two-dimensional textures of the same size as the first two.

Finally, since by rearranging Equation 6.7, we obtain

$$\mathcal{M}f^{(n+1)}_{VMLS}(\mathbf{x}) = \sum_{i=1}^{N} \left[f_i + \sum_{j=0}^{n} \left(f_i - \mathcal{M}f^{(j)}_{VMLS}(\mathbf{x}_i) \right) \right] \varphi_i(\mathbf{x}),$$

it is possible to accumulate the results of the iterative process at each vertex as

$$g(\mathbf{x}_i) = f_i + \sum_{j=0}^{n} \left(f_i - \mathcal{M}f^{(j)}_{VMLS}(\mathbf{x}_i) \right), \tag{6.10}$$

and store them in the texture holding the data points instead of the scalar values f_i. During rendering, the reconstructed value at the evaluation point $\mathbf{x}$ is simply calculated as

$$\mathcal{M}f^{(n+1)}_{VMLS}(\mathbf{x}) = \sum_{i=1}^{N} g(\mathbf{x}_i)\varphi_i(\mathbf{x}).$$

Similar arguments are used for the gradient vector.

Data structure. A Kd-tree subdivision is used to generate a partition of the space so as to limit, to an upper bound, the number of vertices of the mesh fetched to evaluate the function at a given position. The idea was to limit in turn the number of vertex supports intersecting a leaf node, including the supports completely contained in the node. Although using grids of linked lists, as is usual done in GPU implementations for which proximity queries are required, is a better choice in terms of performance, to ensure this upper limit a very fine grid would have to be constructed when dealing with highly anisotropic meshes where the cell sizes vary considerably. For instance, the Blunt Fin dataset is a well known dataset which does not impose a challenge to currently available rendering methods. However, meshless methods suffer from its particularly stretched cells [75]. In the tests, for this dataset, the kd-tree partition needed about 3000 leaf nodes to ensure an upper limit of 250 supports intersecting a single leaf, while the grid of linked lists needed $2k \times 1.5k \times 5k$ cells.

The construction of the Kd-tree was performed as usual by subdividing the nodes at a position determined by means of a bisection process so as to minimize the difference between the number of intersecting supports at each child node, *i.e.*, to approach the ratio of supports in each child to 1. The direction of the normal vector to the dividing plane is chosen along the axis for which the best ratio is obtained. The process stops after a user-defined number of iterations. Since traversing a Kd-tree on the GPU is computationally expensive, it was decided to render each leaf node in a different render pass and blend the result of each leaf node using alpha blending, as explained in the following.

Render passes. To handle mesh boundaries different from the bounding box of the volume, two initial render passes are performed every time the viewing vector is changed. The first render pass stores the intersection of each ray with the front faces of the surface mesh in a texture bound to a `framebuffer object`. In the second render pass the same operation is performed for the back faces. A single `framebuffer object` can be used for the two texture attachments to reduce the number of `frambebuffer` bindings. A limitation of this approach is that the meshes must be convex.

These two textures are then used in the render passes performed for rendering each leaf node. For this, the textures containing the vertex positions, gradients, volumetric data and transformation matrices are input to each render pass. Since each leaf has more than one support intersecting it, a list of vertices whose supports intersect the leaf is constructed. The list contains the texture coordinates of the actual information of each vertex stored in the previously mentioned textures. All lists are then stored in a further two-dimensional texture to be accessed in the fragment program. To access this list, during each render pass, the texture coordinates of the first position of the list in the texture is passed as an uniform variable, together with the number of vertices in the list. With this information, and the entry and exit points of the ray in the leaf, computed in the vertex and fragment programs respectively as explained below, the ray in the fragment program is traversed calculating the reconstructed value as in Equation 6.10. The entry point of the ray in the leaf is calculated by rendering the front faces of the bounding box of the leaf node and interpolating the positions of the vertices at each fragment. With the ray direction, given by the normalized vector from the camera position to the entry point, the exit point is calculated in the fragment program using the information of the planes tangent to the back faces of bounding box of the leaf node (recall that a leaf node has an axis aligned bounding box). The entry and exit points of the ray in the bounding box of the leaf node are compared to the entry and exit points of the ray in the mesh calculated as described above, so that the sampling is ensured to be performed inside the mesh at all times. In order to obtain a correct forward blending of the results of each leaf node, the nodes are rendered in order of proximity to the camera position.

The convergence to radial basis functions interpolation of the method described above was confirmed with numerical tests where the reconstructed value at the vertices of the mesh using iterated approximate approximation was compared to the input data stored in a curvilinear mesh with highly stretched cells. The data was generated by sampling three different functions at the vertices of the grid, namely,

$$\begin{aligned}
f_1(x,y,z) &= \frac{(1.25+\cos(5.4y)(\cos(6z)))}{(6+6(3x-1)^2)}. \\
f_2(x,y,z) &= \frac{(\tanh(9z-9x-9y)+1)}{9}, \\
f_3(x,y,z) &= 0.75\exp\left[-\frac{(9x-2)^2+(9y-2)^2+(9z-2)^2}{4}\right] \\
&+ 0.75\exp\left[-\frac{(9x+1)^2}{49}-\frac{(9y+1)^2}{10}-\frac{(9z+1)^2}{10}\right] \\
&+ 0.5\exp\left[-\frac{(9x-7)^2+(9y-3)^2+(9z-5)^2}{4}\right] \\
&- 0.2\exp\left[-(9x-4)^2-(9y-7)^2-(9z-5)^2\right],
\end{aligned}$$

The domain of these functions is $C = [0,1] \times [0,1] \times [0,1]$ and their ranges are $[-0.37, 0, 37]$, $[0.0, 0.22]$ and $[-0, 1, 1.1]$ respectively. After an average of 50 iterations the method converged and mean and maximum errors of $8E-9$ and $5E-8$, respectively, were obtained. It is worth to mention that no divergence was obtained in the tests. On the other hand, the convergence is slow and, as stated by Fasshauer and Zhang [48], by testing new values of ϵ it can be possible to reach convergence faster, but divergence can occur. However, since the results of the iterated process are pre-computed, this slow convergence did not have any impact in the processing time during rendering.

Regarding performance, tests using an Nvidia GeForce 8800 Ultra graphics card were carried out. The main difficulty in the implementation of the method on the GPU was finding an effective way to deal with proximity queries. The use of a Kd-tree allowed us to limit the number of vertices visited for reconstructing the function at a given point, but the performance was dramatically decreased by the

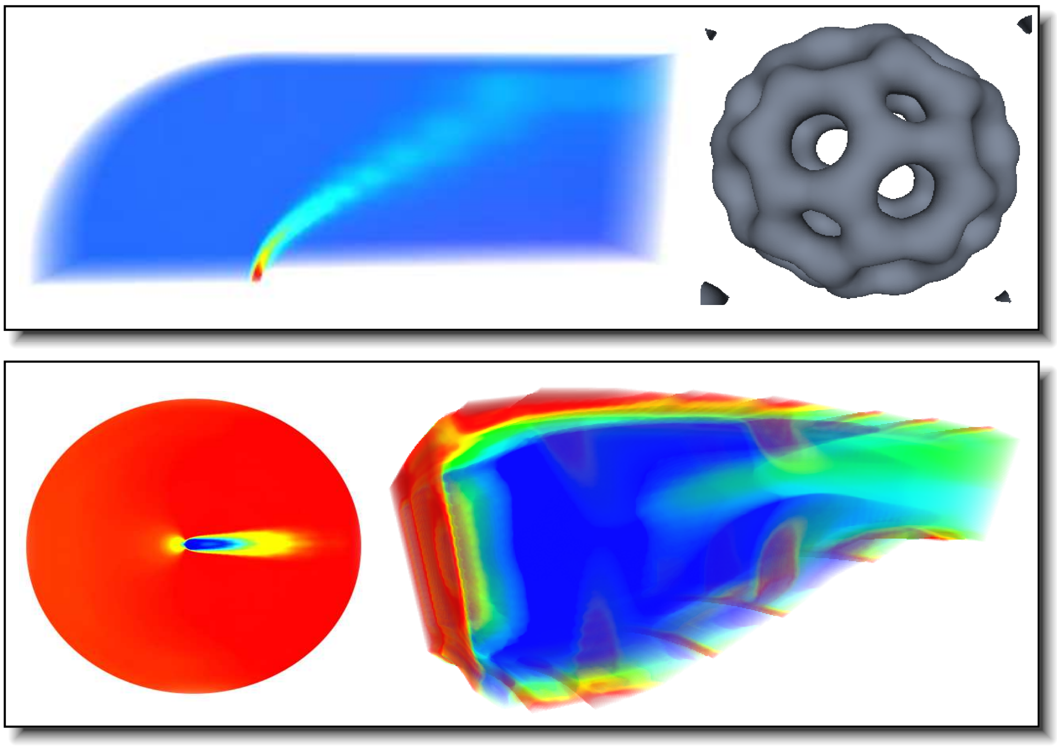


Figure 6.6: Renderings of the Blunt Fin, Bucky Ball, Oxygen Post, and Combustion Chamber datasets (see color plates).

large number of render passes. Implementing the method in a single render pass with a cleverer approach is a task that must be addressed in the future. Thus, for the Blunt Fin ($40 \times 32 \times 32$ cells), Fighter (70125 tetrahedra) and Oxygen Post ($38 \times 76 \times 38$ cells) the processing time was 2.95s, 2.97s and 4.45s per frame respectively. On the other hand, the performance with the Heat Sink (121668 tetrahedra), Nucleon ($41 \times 41 \times 41$ cells), and Combustion Chamber ($57 \times 33 \times 25$ cells) datasets decreased and these datasets were rendered in 11.64s, 11.72s and 10.09s respectively.

Visual rendering results are shown in Figure 6.6 with different datasets. In all cases, including the numerical tests reported above, a fill size $h = 0.25$ and scaling parameter $\epsilon = 0.9$. These values provided good results both visually and in terms of accuracy. This is a further advantage of the method presented compared to approaches where parameter tunning is needed for each dataset.

Although the method proposed presents convergence in all the test cases, it is necessary to prove similar results to the presented by Fasshauer and Zhang [48], in order to demonstrate the convergence to an ellipsoidal basis function. Numerical tests regarding the dependence of the quality of the solution with respect to the cell size, cell aspect ratio and frequency of the test function sampled at different locations in the domain must be also carried out.

6.4 Moving Least-Squares Volume Deformation

Volume deformation is used for a wide range of applications. While physically-based methods yield a plausible deformation, in applications such as volume registration and volume exploration the interest focuses on providing a fast and easy to manipulate means for modeling volumes. Deformation algorithms for different types of grids based on a combination of atomic transformations, such as scale, twist, squeeze, taper or bend have been proposed. Although these transformations are easy to use separately, a combination of them to generate complex deformations is a difficult problem. Moreover, most of them are not suitable for direct manipulation by the user even when they can be controlled using few parameters.

A meshless approach was proposed by Müller *et al.* [119] based on the idea of performing the registration of two point sets, where a polar decomposition of a quadratical matrix is used. Interactive deformation of volumetric data is also addressed in the work by Chen *et al.* [31] and by Westermann *et al.* [170] by using free-form deformations. An hyperpatch with 64 control points is used to deform the enclosing space and thus the object. Gibson [57] proposed the 3D-Chainmail algorithm, which is based on the propagation of the deformation at the vertices of the grid, where the deformation of a vertex is based on the deformation of its neighbors.

Deformation based on the idea of providing an as-rigid-as-possible deformation has gained popularity in the last years. The term as-rigid-as-possible was introduced by Alexa *et al.* [6] to describe a deformation for which the scaling and shearing are minimal. This yields a natural and plausible deformation, as has been shown by Schaefer *et al.* [136] and Igarashi *et al.* [74] who presented deformation algorithms for bidimensional images. The advantage of the method presented by Schaefer *et al.* is that, by using moving least-squares, it is not necessary that the image be triangulated and that a closed-formula for the deformation can be obtained. An efficient and effective extension of this method to three dimensions was proposed by Cuno *et al.* [37] which is the basis of the work presented in this section. Cuno's approach was chosen to solve the minimization problem because of its advantages compared to other methods which will be discussed later in this section, when a description of how the minimization problem can be solved using orthogonal matrices and quaternions is given.

Since the goal of the method presented here is to aid in the process of volume exploration, the aim was to provide a means to deform the volume that permits the use of known hardware-accelerated volume rendering methods. As mentioned before, Cuno's generalization to three dimensions of Schaefer's moving least-squares deformations was used. Although Cuno's method was proposed for surface meshes, the mathematical background can be applied to three-dimensional meshes straightforwardly. Since the goal is to provide a means to deform any type of volume, the approach used is based on calculating a displacement map to obtain undeformed positions from deformed ones, as in the work by Rezk-Salama [131]. This is the reason why the GPU implementation presented, as will be seen later, focuses on creating a Cartesian grid with displacement vectors, which can be used to recover the position given by the inverse map of the deformation. Therefore, a backward mapping is defined in contrast to the forward mapping defined by Cuno.

One important contribution presented here is the inclusion of nonlinear polynomial transformations in the set of moving least-squares deformations originally proposed by Schaefer and extended to three dimensions by Cuno, which comprises affine, similarity and rigid transformations. This way, a further set of polynomial transformations is available, which complies with the requirement of being represented by a closed formula. To that end, orthogonal polynomials are used in order to be able to define a moving least-squares polynomial approximation free of systems of equations. Nonlinear polynomial transformations are able to provide deformations, such as bending, that cannot be modeled with linear transformations. The GPU implementation of this polynomial transformations has shown interactive frame rates as will be discussed in the results. It is also shown how cuts can be simulated and the special handling needed to use them in a moving least-squares deformation is described.

6.4.1 Affine, similarity and rigid deformations

In this section, moving least-squares deformations in three dimensions for backward mapping are described, which are obtained from the forward mapping version [37] straightforwardly by means of variable exchange.

Moving least-squares deformation uses control points to let the user manipulate the volume. The set of N control points $\{\mathbf{p}_i\}$ and their deformed positions $\{\mathbf{q}_i\}$ is the only input to the method. A function $\mathcal{M}f_{DMLS}$ is then approximated that maps any point $\mathbf{u}$ in the undeformed volume to a point $\mathbf{v}$ in the deformed volume. The function $\mathcal{M}f_{DMLS}$ is a continuous interpolating function, *i.e.*, $\mathcal{M}f_{DMLS}(\mathbf{p}_i) = \mathbf{q}_i$, which holds that if $\forall \mathbf{p}_i = \mathbf{q}_i$ then $\mathcal{M}f_{DMLS}(\mathbf{p}_i) = \mathbf{p}_i$; $i = 1 \cdots N$.

By using moving least-squares to find $\mathcal{M}f_{DMLS}$, Schaefer computes a different transformation $l_{\mathbf{u}}$ for each point $\mathbf{u}$ by solving

$$\min \sum_{i=1}^{N} \omega_i (l_{\mathbf{u}}(\mathbf{p}_i) - \mathbf{q}_i)^2. \tag{6.11}$$

The weighting function used in the moving least-squares approximation evaluated in $\mathbf{p}_i$ and $\mathbf{u}$ is denoted here, differently from previous sections, by ω_i, and defined as

$$\omega_i = \frac{1}{(\mathbf{p}_i - \mathbf{u})^{2\alpha}}$$

and $\alpha = h^2$, where h is the fill size. The function $\mathcal{M}f_{DMLS}$ that minimizes this expression provides a forward mapping. However, as stated before, in order to better fit the deformation method to rendering algorithms such as ray-casting, the aim is to find the backwards transformation $l_{\mathbf{v}}^{-1}$ that maps each deformed position $\mathbf{v}$ to its position $\mathbf{u}$ in the undeformed volume. This is easily accomplished by formulating the minimization as

$$\min \sum_{i=1}^{N} \omega_i'(\mathbf{p}_i - l_{\mathbf{v}}^{-1}(\mathbf{q}_i))^2, \tag{6.12}$$

where

$$\omega_i' = \frac{1}{(\mathbf{q}_i - \mathbf{v})^{2\alpha}}.$$

This is an advantage of moving least-squares deformations compared to, for instance, free-form deformations, where the inverse transformation is not so simple to find. Thus, the deformation $\mathcal{M}f_{DMLS}$ is defined as

$$\mathcal{M}f_{DMLS}(\mathbf{v}) = l_{\mathbf{v}}^{-1}(\mathbf{v}).$$

Since $l_{\mathbf{v}}^{-1}$ is an affine transformation, it can be written as

$$l_{\mathbf{v}}^{-1}(\mathbf{x}) = \mathbf{x}\mathbf{M} + \mathbf{t}, \tag{6.13}$$

where $\mathbf{M}$ and $\mathbf{t}$ describe a rotation and a translation respectively. As done by Schaefer, $\mathbf{t}$ can be written as

$$\mathbf{t} = \mathbf{p}_* - \mathbf{q}_*\mathbf{M},$$

where $\mathbf{p}_*$ and $\mathbf{q}_*$ are the weighted centers of mass of the manipulation points

$$\mathbf{p}_* = \frac{\sum_{i=1}^{N} \omega_i'\mathbf{p}_i}{\sum_{i=1}^{N} \omega_i'} \quad \mathbf{q}_* = \frac{\sum_{i=1}^{N} \omega_i'\mathbf{q}_i}{\sum_{i=1}^{N} \omega_i'}.$$

Therefore, it is possible to rewrite Equation 6.13 as

$$l_{\mathbf{v}}^{-1}(\mathbf{x}) = (\mathbf{x} - \mathbf{q}_*)\mathbf{M} + \mathbf{p}_*, \tag{6.14}$$

and, by letting $\hat{\mathbf{p}}_i = \mathbf{p}_i - \mathbf{p}_*$ and $\hat{\mathbf{q}}_i = \mathbf{q}_i - \mathbf{q}_*$, the moving least-squares problem from Equation 6.12 becomes

$$\min \sum_{i=1}^{N} \omega_i' \|\hat{\mathbf{q}}_i\mathbf{M} - \hat{\mathbf{p}}_i\|^2. \tag{6.15}$$

The transformation matrix $\mathbf{M}$ determines the behavior of the deformation and is not restricted to affine transformations. Specifically, an as-rigid-as-possible deformation is obtained by restricting $\mathbf{M}$ to represent a rotation. In the following, the backward mapping versions of the affine, rigid and similarity three-dimensional moving least-squares deformations are described. Closed formulas are obtained for these deformations. Thus, as mentioned before, later it is shown how closed formulas for non-linear deformations can be obtained by using orthogonal polynomials.

Affine deformations. If no restriction is imposed on $\mathbf{M}$ in Equation 6.15, the solution is an affine transformation that can be obtained by deriving the equation with respect to $\mathbf{M}$:

$$\begin{aligned}\frac{\partial \sum_{i=1}^{N} \omega_i' \|\hat{\mathbf{q}}_i\mathbf{M} - \hat{\mathbf{p}}_i\|^2}{\partial \mathbf{M}} &= \frac{\partial \sum_{i=1}^{N} \omega_i' (\hat{\mathbf{q}}_i\mathbf{M} - \hat{\mathbf{p}}_i)(\hat{\mathbf{q}}_i\mathbf{M} - \hat{\mathbf{p}}_i)^T}{\partial \mathbf{M}} \\ &= \frac{\partial \sum_{i=1}^{N} \omega_i' \left(\hat{\mathbf{q}}_i\mathbf{M}\mathbf{M}^T\hat{\mathbf{q}}_i - 2\hat{\mathbf{p}}_i\mathbf{M}^T\hat{\mathbf{q}}_i^T + \hat{\mathbf{p}}_i\hat{\mathbf{p}}_i^T\right)}{\partial \mathbf{M}} \\ &= 2\sum_{i=1}^{N} \omega_i' \left(\hat{\mathbf{q}}_i^T\hat{\mathbf{q}}_i\right)\mathbf{M} - 2\sum_{i=1}^{N} \omega_i'\hat{\mathbf{q}}_i^T\hat{\mathbf{p}}_i.\end{aligned}$$

The root of this equation gives the transformation matrix

$$\mathbf{M} = \left(\sum_{i=1}^{N} \omega_i'\hat{\mathbf{q}}_i^T\hat{\mathbf{q}}_i\right)^{-1} \sum_{i=1}^{N} \omega_i'\hat{\mathbf{q}}_i^T\hat{\mathbf{p}}_i. \tag{6.16}$$

Since this requires the inversion of a 3×3 matrix, for which analytic solutions exist, Equation 6.14 can be considered a closed formula. This solution is exactly the same as the one presented by Schaefer and collaborators for the two-dimensional case. Only $\mathbf{q}_i$ and $\mathbf{p}_i$ where interchanged. However, in this case, no pre-processing is possible as in the method by Schaefer, since the points $\mathbf{q}_i$ are not fixed. It is important to note that the control points cannot be coplanar. Otherwise the matrix becomes singular and the deformation function is undefined. From this, the minimum number of control points (4) follows.

Rigid deformations. As done by Schaefer, to obtain a rigid deformation it is necessary to restrict $\mathbf{M}$ to a rotation matrix, *i.e.*, $\mathbf{M} \in SO_3(\mathbb{R})$, where $SO_3(\mathbb{R})$ is the group of real orthogonal 3×3 matrices, with the property $\forall \mathbf{M} \in SO_3(\mathbb{R})$, $\det \mathbf{M} = 1$. With this restriction, the optimization problem becomes

$$\min_{\mathbf{M}\in SO_3(\mathbb{R})} \sum_{i=1}^{N} \omega_i' \|\hat{\mathbf{q}}_i\mathbf{M} - \hat{\mathbf{p}}_i\|^2. \tag{6.17}$$

The formulation by Schaefer for the two-dimensional case restricts the matrix to be a 2×2 orthogonal matrix, *i.e.*, $\mathbf{M}^T\mathbf{M} = \mathbf{I}$. In two dimensions this suffices to restrict the solution to a rotation, since in this case $\det \mathbf{M} = 1$ always holds. For the three-dimensional case this is however not enough since 3×3 orthogonal matrices can have determinant ± 1 and therefore a rotation as well as a mirroring can be represented by such matrices. Independently from the procedure to solve the problem, a deformation that is not as-rigid-as-possible is obtained when $\det \mathbf{M} = -1$. Therefore, the matrix must be restricted to $\mathbf{M}^T\mathbf{M} = \mathbf{I}$ and $\det \mathbf{M} = 1$. Alternatively it is possible to use another representation such as quaternions or rotation angle/axis. This latter approach is used by Cuno for surface meshes and is applied here to compute the displacement map as will be described in the following.

To solve the moving least-squares problem of Equation 6.17, it is rewritten as a maximization problem that only involves matrix and vector multiplications. For the rotation matrix a new representation of the rotation by means of rotation axis and angle is used, which makes it easier to solve the optimization problem. The estimation of the rotation axis and angle is carried out as an eigenvalue-finding problem that is solved by finding the roots of a polynomial of degree 4. For this, Equation 6.17 is rewritten as

$$\min_{\mathbf{M} \in SO_3(\mathbb{R})} \left(-2 \sum_{i=1}^{N} \omega'_i \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T + \sum_{i=1}^{N} \omega'_i \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T + \sum_{i=1}^{N} \omega'_i \hat{\mathbf{q}}_i \mathbf{M}\mathbf{M}^T \hat{\mathbf{q}}_i^T \right).$$

Note that in this equation $\mathbf{M}\mathbf{M}^T = \mathbf{I}$ and therefore the last two terms are constant and can be disregarded in the minimization problem. Since the first term is negative, the problem can be rewritten as the maximization problem

$$\max_{\mathbf{M} \in SO_3(\mathbb{R})} \sum_{i=1}^{N} \omega'_i \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T . \tag{6.18}$$

Three-dimensional rotation. In the three-dimensional space it is possible to represent a rotation by using matrices, quaternions, Euler angle or rotation axis and rotation angle. In order to find a rotation matrix that solves Equation 6.18 it would be necessary to find nine unknowns. Therefore, a representation that uses fewer variables, specifically a rotation axis $\mathbf{e}$ and a rotation angle α is used. The rotation matrix depends on vector $\mathbf{e}$ and scalar α as

$$\mathbf{M} = \mathbf{e}^T\mathbf{e} + \cos(\alpha)(\mathbf{I} - \mathbf{e}^T\mathbf{e}) + \sin(\alpha) \begin{pmatrix} 0 & \mathbf{e}_z & -\mathbf{e}_y \\ -\mathbf{e}_z & 0 & \mathbf{e}_x \\ \mathbf{e}_y & -\mathbf{e}_x & 0 \end{pmatrix}. \tag{6.19}$$

By replacing this in Equation 6.18, we obtain

$$\max_{\|e\|=1,\cos(\alpha)^2+\sin(\alpha)^2=1} \left(\mathbf{eCe}^T + \cos(\alpha)(s - \mathbf{eCe}^T\right) + \sin(\alpha)\mathbf{ke}^T), \tag{6.20}$$

where

$$\mathbf{C} = \sum_{i=1}^{N} \omega_i' \hat{\mathbf{p}}_i^T \hat{\mathbf{q}}_i, \tag{6.21}$$

$s = \text{trace}(\mathbf{C})$ and $\mathbf{k} = \sum_{i=1}^{N} \omega_i' \hat{\mathbf{p}}_i \times \hat{\mathbf{q}}_i$. The first restriction in the maximization problem given by Equation 6.20 makes the vector $\mathbf{e}$ be normalized. The second restriction is apparently unnecessary since $\cos(\alpha)^2 + \sin(\alpha)^2 = 1$ is always true. However, this restriction is used to solve Equation 6.20 for $\sin(\alpha)$ and $\cos(\alpha)$ that hold this restriction. For this, the Lagrange function

$$\begin{aligned} L(\mathbf{e}, \sin(\alpha), \cos(\alpha); \lambda_1, \lambda_2) &= \mathbf{eCe}^T + \cos(\alpha)(s - \mathbf{eCe}^T) + \sin(\alpha)\mathbf{ke}^T + \\ &\quad \lambda_1(1 - \|\mathbf{e}\|) + \\ &\quad \lambda_2(1 - \cos(\alpha)^2 - \sin(\alpha)^2) \end{aligned}$$

is used. Here λ_1 and λ_2 are the Lagrange multipliers. The stationary points of L are obtained as the roots of the first partial derivatives of L with respect to $\mathbf{e}$, $\cos(\alpha)$ and $\sin(\alpha)$:

$$(1 - \cos(\alpha))\mathbf{e}(\mathbf{C} + \mathbf{C}^T) + \sin(\alpha)\mathbf{k} = \lambda_1 \mathbf{e} \tag{6.22}$$

$$s - \mathbf{eCe}^T = 2\lambda_2 \cos(\alpha) \tag{6.23}$$

$$\mathbf{ke}^T = 2\lambda_2 \sin(\alpha). \tag{6.24}$$

From Equation 6.24, $\sin(\alpha) = \frac{\mathbf{ke}^T}{2\lambda_2}$, which replaced in Equation 6.22 gives

$$\mathbf{e}(\mathbf{C} + \mathbf{C}^T) + \frac{1}{2\lambda_2(1 - \cos(\alpha))}\mathbf{ke}^T\mathbf{k} = \frac{\lambda_1}{1 - \cos(\alpha)}\mathbf{e}. \tag{6.25}$$

By letting

$$\begin{aligned} \mathbf{N} &= \mathbf{C} + \mathbf{C}^T \\ a &= \frac{1}{2\lambda_2(1 - \cos(\alpha))} \\ \lambda &= \frac{\lambda_1}{1 - \cos(\alpha)} \end{aligned} \tag{6.26}$$

it is possible to rewrite Equation 6.25 as

$$\mathbf{e}(\mathbf{N} + a\mathbf{k}^T\mathbf{k}) = \lambda \mathbf{e}, \tag{6.27}$$

which means that the rotation axis $\mathbf{e}$ is an eigenvector of the matrix $(\mathbf{N} + a\mathbf{k}^T\mathbf{k})$ corresponding to the eigenvalue λ. Since λ is a root of the characteristic polynomial $P(\lambda)$ of the matrix $(\mathbf{N} + a\mathbf{k}^T\mathbf{k})$, using the Frobenius norm $\|\cdot\|_F$ and the approximation

$$\det(\mathbf{I} + \mathbf{A}) \approx 1 + \operatorname{trace}(\mathbf{A}),$$

we have

$$\begin{aligned} P(\lambda) &\approx \lambda^3 - \lambda^2 \left[\operatorname{trace}(\mathbf{N}) + a\right] \mathbf{k}\mathbf{k}^T \\ &+\lambda \left[\frac{1}{2}(\operatorname{trace}(\mathbf{N})^2 - \|\mathbf{N}\|_F^2) + a(\mathbf{k}\mathbf{k}^T \operatorname{trace}(\mathbf{N}) - \mathbf{k}\mathbf{N}\mathbf{k}^T)\right] \\ &- \det(\mathbf{N})(1 + \mathbf{k}\mathbf{N}^{-1}\mathbf{k}^T a). \end{aligned}$$

The variable a in $P(\lambda)$ is unknown, since it depends on λ_2. It is possible to show that a and λ are related. For that, $\mathbf{e}^T$ is multiplied to both sides of Equation 6.22 and by using Equations 6.23 and 6.24,

$$2(1 - \cos\alpha)(s - 2\lambda_2 \cos\alpha) + 2\lambda_2 \sin\alpha \sin\alpha = \lambda_1 \mathbf{e}\mathbf{e}^T$$

is obtained. By rearranging this equation, noting that $\mathbf{e}\mathbf{e}^T = 1$ and using the Definition 6.26, we obtain

$$a = \frac{1}{\lambda - 2s}. \tag{6.28}$$

Thus, the equation $P(\lambda) = 0$ becomes

$$\begin{aligned} 0 &= \lambda^4 - \lambda^3 4s + \lambda^2 \left[6s^2 - 2\|\mathbf{C}\|_F^2\right] + \\ &\quad \lambda \left[4\left(\|\mathbf{C}\|_F^2 - s^2\right)s - 2\mathbf{k}\mathbf{C}\mathbf{k}^T - \det(\mathbf{N})\right] + \\ &\quad \det(\mathbf{N})(2s - \mathbf{k}\mathbf{N}^{-1}\mathbf{k}^T). \end{aligned}$$

To find the root of this polynomial, it is converted to a depressed quartic function by replacing $y = \lambda - s$:

$$\begin{aligned} 0 &= y^4 - \\ &\quad y^2\left(2\|\mathbf{C}\|_F^2\right) - \\ &\quad y\left(8\det(\mathbf{C})\right) + \\ &\quad \det(\mathbf{N})(2s - \mathbf{k}\mathbf{N}^{-1}\mathbf{k}^T) - 8\det(\mathbf{C})s + 2\|\mathbf{C}\|_F^2 s^2 - s^4, \end{aligned}$$

which can be solved using the method by Ferrari. Since the problem at hand is a maximization problem (Equation 6.18), the largest root r_{max} of this quartic function gives the solution.

Determining the rotation axis and angle. Given the maximal root of the characteristic polynomial, by substituting $y = \lambda - s$ and using Equation 6.28, the rotation

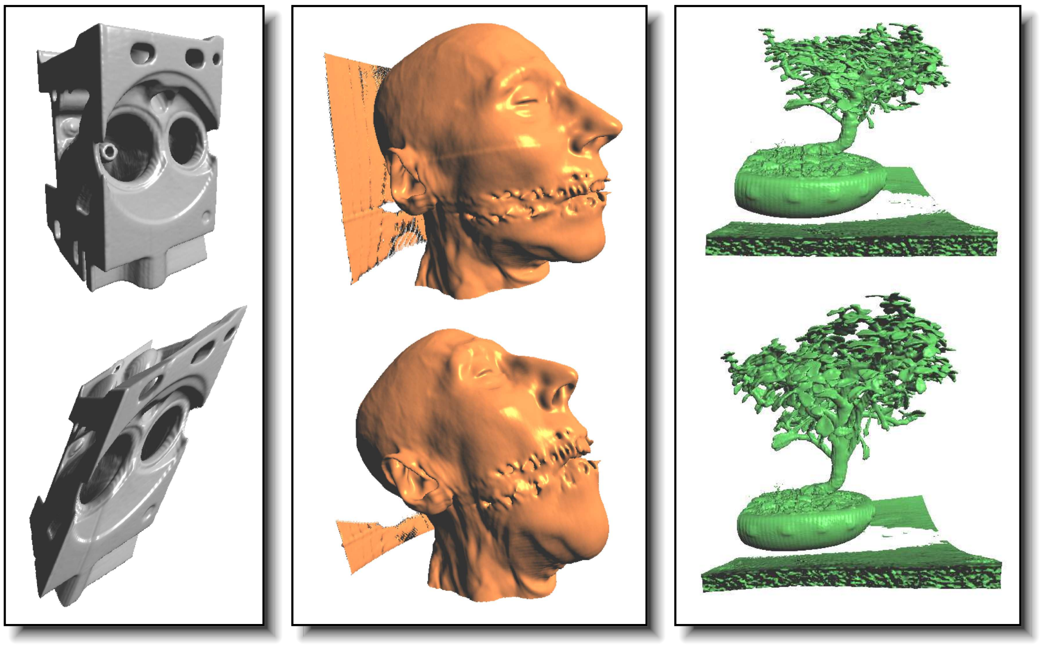


Figure 6.7: Examples of affine (left), rigid (center) and similarity (right) moving least-squares deformations. The top row shows the undeformed object, whereas the bottom row shows the corresponding deformation.

axis $\mathbf{e}$ can be obtained as an eigenvector of $(\mathbf{N} + a\mathbf{k}^T\mathbf{k})$ corresponding to the now known eigenvalue λ. For small deformations, this approach can be problematic since a can be very large. By substituting $\varpi = -a\mathbf{k}\mathbf{e}^T$ and $\mathbf{u} = \frac{\mathbf{e}}{\varpi}$ it is possible to rewrite Equation 6.27 as

$$\mathbf{u}(\mathbf{N} - \lambda\mathbf{I}) = \mathbf{k}. \tag{6.29}$$

With this, $\mathbf{e}$ is obtained by normalizing $\mathbf{u}$, which is obtained by means of a matrix inversion. To obtain the rotation angle α, the optimization problem given by Equation 6.20 is solved with respect to $\sin(\alpha)$ and $\cos(\alpha)$. From Equations 6.26 and 6.28 we have

$$\lambda - 2s = 2\lambda_2(1 - \cos(\alpha)).$$

By adding Equation 6.23 we obtain, for λ_2,

$$2\lambda_2 = \lambda - s - \mathbf{e}\mathbf{C}\mathbf{e}^T.$$

Therefore, from Equations 6.23 and 6.24 we obtain

$$\cos(\alpha) = \frac{s - \mathbf{e}\mathbf{C}\mathbf{e}^T}{\lambda - s - \mathbf{e}\mathbf{C}\mathbf{e}^T} \quad \text{and} \quad \sin(\alpha) = \frac{\mathbf{k}\mathbf{e}^T}{\lambda - s - \mathbf{e}\mathbf{C}\mathbf{e}^T}.$$

Similarity deformations. A generalization of the rigid deformations is the similarity deformation, which consists of a rotation and an uniform scaling. By introducing a scaling factor $\mu_s \in \mathbb{R}$ into Equation 6.12 a new minimization problem can be stated as

$$\min_{\mathbf{M}\in SO_3(\mathbb{R})} \sum_{i=1}^{N} \omega_i' \|\mu_s \hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i\|.$$

This can also be restated as a maximization problem as

$$\max_{\mathbf{M}\in SO_3(\mathbb{R})} 2\mu_s \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T - \mu_s^2 \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \hat{\mathbf{q}}_i^T.$$

By assembling the Lagrange functions and deriving with respect to μ_s the optimality condition

$$\sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T - \mu_s \sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \hat{\mathbf{q}}_i^T = 0$$

is obtained. Since the optimality condition of Equation 6.20 states that solving $\sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \mathbf{M} \hat{\mathbf{p}}_i^T = r_{max}$ suffices, we obtain

$$\mu_s = \frac{r_{max}}{\sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i \hat{\mathbf{q}}_i^T}.$$

The matrix $\mathbf{M}$ is calculated as in the case of rigid deformations.

6.4.2 Nonlinear polynomial deformation

As mentioned before, nonlinear polynomial transformations are able to provide deformations, such as bending, that cannot be modeled with linear transformations. Non-linear moving least-squares deformations can be easily obtained by solving Equation 6.12 with the function $l_{\mathbf{v}}^{-1}(\mathbf{x})$ being a polynomial of arbitrary degree. Let $\Psi = \{\psi_1, \ldots, \psi_M\}$, where ψ_j are basis functions (polynomial functions in this case) and $\mathcal{F} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\} \subsetneq \mathbb{R}$. Let also $\Psi_j = [\psi_j(\mathbf{q}_1), \ldots, \psi_j(\mathbf{q}_N)]$, $\Gamma = [\mathbf{p}_1, \ldots, \mathbf{p}_N]$ and define the inner product $\langle , \rangle_{\omega'} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$ as a weighted sum:

$$\langle \xi, \eta \rangle_{\omega'} = \sum_{i=1}^{N} \xi_i \eta_i \omega_i', \tag{6.30}$$

Then, the new minimization problem can be formulated as

$$\min \sum_{i=1}^{N} \omega_i' (\mathbf{p}_i - \varrho_{\mathbf{v}}^{-1}(\mathbf{q}_i))^2, \tag{6.31}$$

where

$$\varrho_{\mathbf{v}}^{-1}(\mathbf{v}) = \sum_{j=1}^{M} c_j(\mathbf{v}) \psi_j(\mathbf{v})$$

and c_j are the unknown coefficients to be found. Note that Ψ must not contain constant terms since the translation $\mathbf{t}$ is not directly computed (see Equation 6.14). These coefficients can be obtained by solving the corresponding normal equation given by

$$\left\{ \sum_{j=1}^{M} \langle \Psi_k, \Psi_j \rangle_{\omega'} c_j = \langle \Gamma, \Psi_k \rangle_{\omega'};\ k = 1, \ldots, M \ . \right. \tag{6.32}$$

Solving this problem would mean having to invert, for instance, for a complete quadratic polynomial, a 10×10-matrix and for a cubic polynomial a 20×20-matrix. The interest here lies on providing a closed formulation of the polynomial transformation of arbitrary degree. This can be achieved by means of multi-variate orthogonal polynomials described in Chapter 3.

Thus, a set Ψ is defined, as before using some orthogonalization process, such that the inner product satisfies $\langle \Psi_i, \Psi_j \rangle_{\omega'} = \kappa_{ij}\delta_{ij}$, where δ_{ij} is the Kronecker delta, System 6.32 becomes a linear system where the coefficient matrix is diagonal. Thus, the approximation is given by the sum

$$\varrho_{\mathbf{v}}^{-1}(\mathbf{x}) = \sum_{j=1}^{M} \psi_j(\mathbf{x}) \frac{\langle \Gamma, \Psi_j \rangle_{\omega'}}{\langle \Psi_j, \Psi_j \rangle_{\omega'}}. \tag{6.33}$$

Again, Ψ must not contain constant terms. The mapping $l_{\mathbf{v}}^{-1}$ is then given by

$$l_{\mathbf{v}}^{-1} = \varrho(\mathbf{x} - \mathbf{q}_*) + \mathbf{p}_*.$$

6.4.3 GPU-based MLS displacement map computation

As stated before, since the goal is to be able to efficiently perform rendering of the deformed volume, GPU-implementations of the transformations described above were developed. The approach followed is based on computing a displacement map $\mathcal{D}$ covering the domain of the volume, which provides the displacement of the backwards mapping given by $l_{\mathbf{v}}^{-1}$. The size of the displacement map can be chosen so as to balance performance and accuracy of the displaced position.

The displacement map is stored in a three-dimensional floating point texture. Linear interpolation is used to obtain the displacement vector at any point in the domain. Thus, in a GPU-based volume renderer, it suffices to fetch the displacement vector corresponding to the current position on the ray from the displacement texture. This is the only change that any volume renderer must suffer to accommodate the deformed volume. However, if gradient information is needed by the volume renderer, some considerations must be taken into account, which are described later.

Since the computation of the displacement map can be performed in parallel and few texture fetches are needed while a significant amount of mathematical

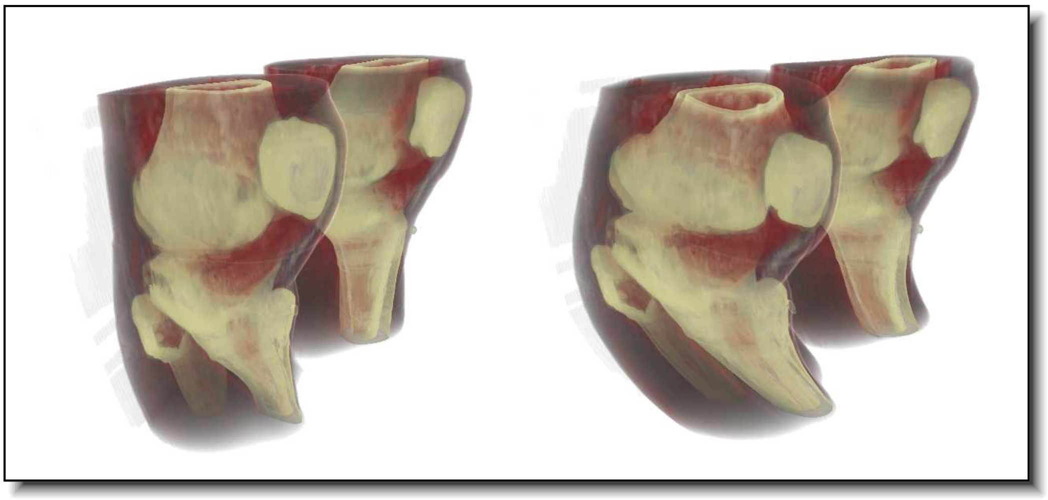


Figure 6.8: Examples of a nonlinear polynomial deformation for the Knee dataset. The bending effect shown is achieved by moving one control point (out of 15) at the bottom of the model.

operations must be performed, this problem is well-suited for a GPU implementation. Besides the performance increase in calculating the deformed positions for each voxel in the displacement map, by calculating it on the GPU no data transfer is needed to upload the map to the graphics hardware memory.

To fill the three-dimensional texture holding the displacement map, the result is rendered into the texture by means of `framebuffer objects`, for which 3D textures are supported by the NVidia GeForce 8 graphics cards. During rendering of the slices of the 3D texture each fragment represents a voxel in the displacement map texture, whose position is given to the fragment shader as texture coordinates. Given the position for the fragment, the shader calculates the undeformed position (backwards mapping) as detailed previously.

To perform this computation, the deformed and undeformed positions of the control points $\mathbf{q}_i$ and $\mathbf{p}_i$, $i = 1, \cdots, N$, respectively, are required. Since N varies and arrays with dynamic size are not supported, a constant equal to N is added dynamically as first line of the shader code. This constant is used throughout the code to define the arrays holding the control points deformed and undeformed positions and partial results obtained during the computation. Then, the shader is compiled in run-time. Since the number of control points doest not change often, the cost of compiling the code in run-time is not relevant.

The fragment shader implementation of the actual algorithms to calculate the undeformed position for each voxel of the displacement texture is straightforward thanks to the capabilities of the latest graphics hardware and is identical to the CPU implementation. One consideration must be pointed out however in the case

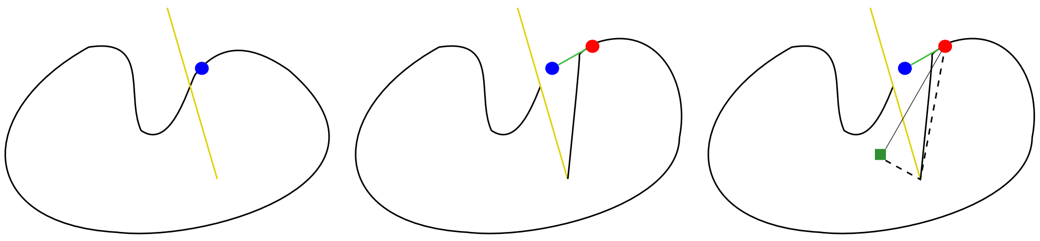


Figure 6.9: A two-dimensional depiction of a volume being cut.

of the non-linear deformations. Since only Shader Model 4 supports more than 32 `TEMP` variables and implementing the Gram-Schmid orthogonalization requires a number of temporal variables depending on the number of control points, the non-linear deformation is only supported by the NVidia GeForce 8 series.

Introducing cuts into moving least-squares deformations. In order to introduce cuts to support the interactive exploration of the volume, the GPU implementation of the deformation algorithms described above is extended. The use of cuts with the rigid transformation provides a plausible behavior and is flexible enough to support different types of cuts on the same model.

A cut can be realized by using a two-dimensional cutting element, *e.g.*, a plane, that divides a portion of the volume in two pieces. Without loosing generality, the following discussion will use half-planes. The behavior of the cut is depicted in Figure 6.9, where a half-plane (represented by the yellow line) slices a volume. The cut is only apparent after a deformation is performed. In the figure it is also possible to see two features of cuts with moving least-squares deformations: one of the manipulation points does not influence the portion of the volume on the other side of the cut, and void regions are created when deforming the volume.

In order to obtain these features for cutting volumes deformed with the moving least-squares approaches described above, a new weight function depending on the position of the point to be deformed relative to the cutting plane is defined. This weight function limits the influence of the control points on the other side of the cutting plane, and is defined as

$$\omega_i' = d(\mathbf{v}, \mathbf{q}_i)\frac{1}{\|\mathbf{q}_i - \mathbf{v}\|^{2\alpha}},$$

where d is a damping function that defines how much influence a displaced control point $\mathbf{q}_i$ has on a voxel $\mathbf{v}$. If $\mathbf{q}_i$ and $\mathbf{v}$ are not occluded one from the other by the cutting plane, then $d(\mathbf{v}, \mathbf{q}_i) = 1$. The function d tends asymptotically to 0 with the size of the coverage. The damping function used in the tests was

$$d(\mathbf{v}, \mathbf{q}_i) = \left(\frac{\|\mathbf{q}_i - \mathbf{v}\|}{\|\mathbf{q}_i - \mathbf{v}\| + \phi(\mathbf{v}, \mathbf{q}_i)}\right)^{\nu},$$

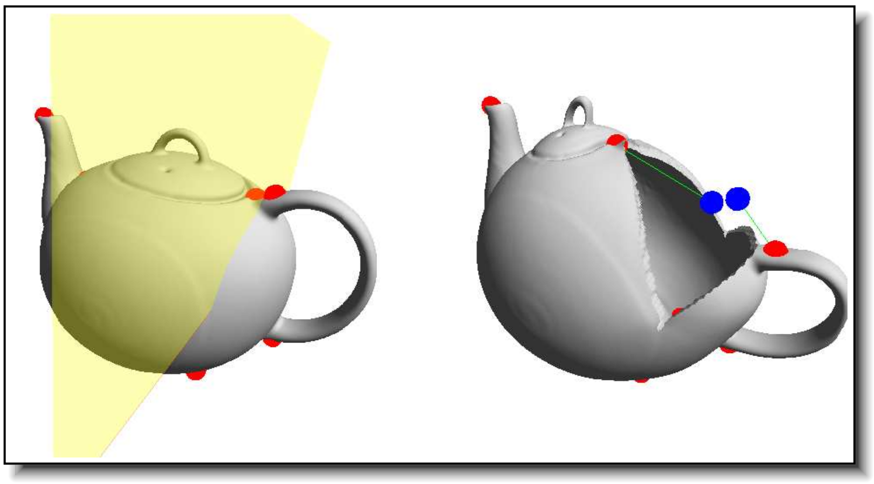


Figure 6.10: Example of a cutting plane in combination with a rigid moving least-squares deformation.

where ϕ returns the length of the shortest indirect path from $\mathbf{q}_i$ to $\mathbf{v}$ that does not cross the cutting plane (Figure 6.9) and the factor $\nu > 1$ makes d decrease more rapidly.

The choice of the damping function d influences directly how the cut will look like. If d is discontinuous, undesired sharp edges are obtained, *e.g.*, during isosurface rendering. Also, ϕ must be strictly monotonically decreasing relatively to the occlusion between $\mathbf{v}$ and $\mathbf{q}_i$, since the influence of a heavily occluded control point must be smaller than that of lighter occluded control points.

To create the void regions it does not suffices to use the damping function d, since this only accounts for the result of the deformation when a cutting plane is present. For this, a test checking if the deformed and the undeformed positions of $\mathbf{v}$ are occluded one from the other by the cutting plane is performed, in which case the deformed position is not taken into account for the rendering.

To visualize the surface of a cut, *e.g.*, during isosurface rendering, the alpha channel of the displacement map is used to store the value 1 if the deformed position of the voxel belongs to a void region. During rendering, the current position on the ray is discarded if the interpolated value of the alpha channel is greater than 0.5. In order to better represent the surface of the cut, adaptive sampling is performed during the rendering with higher frequencies around cut boundaries. The regions around the cut boundaries are easily recognized from the interpolated value in the alpha channel. As one moves forward towards a cut, this interpolated value increases up to 1.

On the left side of Figure 6.7 it is possible to see the characteristic behavior of the affine deformations. The deformed volume presents shears and scales, that

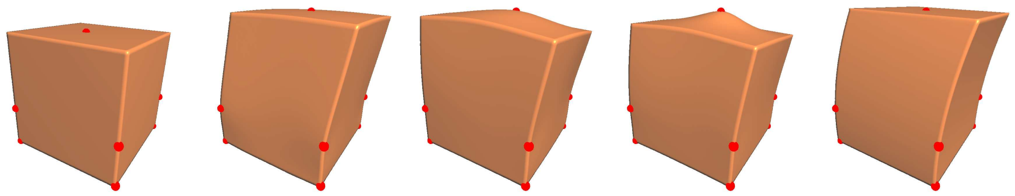


Figure 6.11: Visual comparison of the different moving least-squares deformation methods. From left to right: original model, affine deformation, similarity deformation, rigid deformation and nonlinear polynomial deformation of degree 2. The polynomial deformation shows a bending effect that cannot be accomplished with other deformations.

turns it very dissimilar to the original volume. The effect obtained with these deformations seems very unnatural since most objects in reality are not subject to such transformations. An example of the rigid deformation can also be seen in Figure 6.7. With the deformation it is possible to leave the neck undeformed while pulling the head back. The similarity deformation shown in the figure, on the other hand, depicts clearly the expected scaling.

A visual comparison of the deformations is shown in Figure 6.11. In the figure it is possible to see the characteristics of each transformation. The affine transformation scales the cube, while the similarity transformation increases the size of the top of the cube. The rigid transformation avoids the scaling but the bump on the top is very pronounced as expected. The non-linear deformation (degree 2) shows how bending can be achieved by moving only one manipulation point. This can also be seen in Figure 6.8.

Deforming normal vectors. Volume rendering algorithms need the gradient information, for instance, to render isosurfaces or provide lighting effects for direct volume rendering. After a deformation, the gradient vectors calculated from the data at the undeformed position must be corrected. This can be done using the deformation texture holding the displacement vectors and cut information stored in the alpha channel and an adaptation to backward mapping of the method proposed by Barr [17]. Barr describes the transformation of a normal vector for a forward mapping F as

$$\mathbf{n_q} = \det \mathbf{J}_F \left(\mathbf{J}_F^{-}1\right)^T \mathbf{n_p},$$

where $\mathbf{J}_F$ is the Jacobi matrix of F, $\mathbf{n_p}$ is the normal vector at the undeformed point $\mathbf{p}$ and $\mathbf{n_q}$ is the unknown normal vector at the deformed point $\mathbf{q}$. To adapt this method to backward mapping, let

$$\mathbf{n_q} = \frac{1}{\det \mathbf{J}_G} \left(\mathbf{J}_G\right)^T \mathbf{n_p},$$

where $\mathbf{J}_G$ is the Jacobi matrix of the backward mapping G. It is easy to see that for the Jacobi matrix $\mathbf{J}_D$ of the displacement map, $\mathbf{J}_G = \mathbf{I} + \mathbf{J}_D$, where $\mathbf{I}$ is the identity matrix. Thus,

$$\mathbf{n_q} = \frac{1}{\det \mathbf{I} + \mathbf{J}_D} (\mathbf{I} + \mathbf{J}_D)^T \, \mathbf{n_p}.$$

Since the normal vectors (gradient vectors) for lighting purposes are normalized, the factor $\frac{1}{\det \mathbf{I}+\mathbf{J}_D}$ can be neglected. $\mathbf{J}_D$ can be computed from the displacement texture by means of finite differences. To accelerate the rendering, the matrix $(\mathbf{I} + \mathbf{J}_D)^T$ can be pre-computed and stored in an additional texture. The gradient vector at the cut boundaries must be also recomputed. This vector can be easily calculated as the gradient of the values in the alpha channel of the displacement texture.

In Figure 6.10, a cut obtained with a rigid deformation is depicted. With the cut, it is possible to see the interior of the teapot. The use of manipulation points allows the user to easily change the appearance and nature of the cut. Although implementing the function ϕ on the GPU for a half-plane is straightforward, for more complex geometric cutting objects it could become expensive.

Table 6.2 shows the performance of the deformation algorithms on the CPU and the GPU for different displacement map sizes and number of control points. The tests were performed on a standard PC equipped with a 2GHz processor and 1GB RAM. It is important to note that due to the use of backward mapping, no pre-processing can be applied to accelerate the computation of the displacement map as it was done by Schaefer *et al.* [136] and Cuno *et al.* [37]. As expected, it is possible to see that the processing time is proportional to the size of the displacement texture. The computation of the displacement map is not interactive for medium and large resolutions. On the other hand, the GPU is well suited for this problem as shown in Table 6.2 where it can be seen that interactive frame rates are achieved having a performance increase of a factor of 100 compared to the CPU implementation. The graphics card used was an NVidia GeForce 8800 Ultra. It is important to note that in the case of the GPU implementation, the processing time is not proportional to the size of the displacement texture. In this case, an increase of 8 times the number of voxels in the texture doubles the processing time. By examining other measurements, it is possible to see that the processing time on the GPU is proportional to the number of slices in the 3D textures. This is because the largest part of the processing time is taken by the binding of the current slice to the `framebuffer object`.

6.4.4 Other approaches for moving least-squares deformation

In this section, other potential methods for solving the minimization problem given by Equation 6.12 that are not based on rotation axis and angles are discussed. Disadvantages and limitations will be discussed.

	Deformation algorithm (CPU/GPU)			
Size	affine	rigid	similarity	nonlinear (degree 2)
$64 \times 64 \times 64$	$0.633/0.004$	$1.125/0.008$	$1.133/0.008$	$9.27/0.102$
$128 \times 128 \times 128$	$4.868/0.015$	$8.779/0.045$	$8.976/0.045$	$117.18/0.46$
$256 \times 256 \times 256$	$38.55/0.104$	$69.97/0.325$	$70.55/0.325$	$793.00/0.73$

Table 6.2: Processing time for the CPU and GPU implementations of the deformation methods (in seconds) for displacement maps of different sizes.

By letting $\omega_i' = 1$, the minimization problem

$$\min_{\mathbf{M} \in SO_3(\mathbb{R})} \sum_{i=1}^{N} \|\hat{\mathbf{q}}_i \mathbf{M} - \hat{\mathbf{p}}_i\|^2 \tag{6.34}$$

is obtained, which is a least-squares problem. This problem is named *orthogonal procrustes problem* by Golub and Van Loan [60] and Viklands [160] and is regarded as the problem of fitting two three-dimensional point clouds or as the search for the opposite orientation of two coordinates systems. In general, the orthogonal procrustes problem is formulated for n-dimensional spaces, however in this case it is only meaningful in three dimensions. Analytic solutions to Equation 6.34 are available, for instance, by means of quaternions [71] and orthogonal matrices [70; 13].

Orthogonal matrices. For the orthogonal procrustes problem of Equation 6.34, the method by Golub, based on the singular value decomposition of the unweighted ($\omega_i' = 1$) correlation matrix $\mathbf{C}$ (Equation 6.21), can be used. For any $m \times n$-matrix, the singular value decomposition gives a decomposition of the form $\mathbf{A} = \mathbf{USV}^T$, where the $m \times m$-matrix $\mathbf{U}$ and the $n \times n$-matrix $\mathbf{V}$ are orthogonal while the $m \times n$-matrix $\mathbf{S}$ is diagonal. The rows of $\mathbf{U}$ are the eigenvectors of $\mathbf{AA}^T$ and the columns of $\mathbf{V}$ the eigenvectors of $\mathbf{A}^T\mathbf{A}$. The diagonal elements of $\mathbf{S}$ are the square roots of the eigenvalues of $\mathbf{AA}^T$. In this case, $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{S}$ are 3×3-matrices.

To obtain the rotation matrix that solves Equation 6.34, the singular value decomposition $\mathbf{U_C S_C V_C^T}$ of $\mathbf{C}$ is computed. Then, the rotation matrix is given by $\mathbf{M} = \mathbf{U_C}\mathrm{diag}(1, 1, \det(\mathbf{U_C V_C^T}))\mathbf{V_C^T}$, where $\mathrm{diag}(1, 1, \det(\mathbf{U_C V_C^T}))$ is introduced to ensure $\det(\mathbf{M}) = 1$ in case the $\det(\mathbf{U_C V_C^T}) = -1$. This does not influence the orthogonality of $\mathbf{M}$.

This method does not produce the expected results as can be seen in Figure 6.12. Actually, there are many locations where the deformation obtained is identical to the deformation produced by the method based on rotation angle and axis, but there is nonetheless an abrupt change in the appearance of the deformation, which shows, additionally, noise at the boundaries due to numeric instabili-

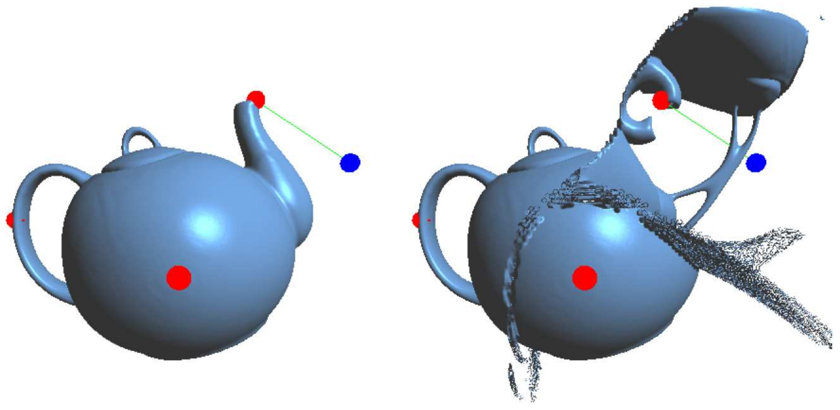


Figure 6.12: Result obtained with orthogonal matrices for the Boston Teapot model.

ties. An explanation for these results is given by Viklands [160], who presented an analysis on the weighted orthogonal procrustes problem, which corresponds to the moving least-squares problem. The weighted orthogonal procrustes problem has, in contrast to the orthogonal procrustes problem, not only one but up to eight minima. Although this number of minima cannot be theoretically proven and was only empirically obtained, it is enough that a single minimum cannot be ensured for the method to be unsuitable for the problem at hand. This approach is very similar to the work by Horn [70] and by Arun *et al.* [13], which are therefore also unsuitable for our problem.

By the polar decomposition, as used by Müller *et al.* [121] for the registration of two point clouds, the matrix $\sum_{i=1}^{N} \omega_i' \hat{\mathbf{q}}_i^T \hat{\mathbf{p}}_i$ is decomposed in the form $\mathbf{RS}$, where $\mathbf{R}$ is a orthogonal matrix and $\mathbf{S}$ is a symmetric matrix. $\mathbf{R}$ is then used as the rotation matrix for the rigid deformation. However, there is a dependency between the polar and the singular value decompositions, which states that $\mathbf{R} = \mathbf{UVT}$ and, therefore, the problem persists.

Quaternions. The method described here for rigid deformations based on finding the rotation axis and angle shows visually no unpleasantness. A search for the optimal rotation by means of quaternions as done by Horn [71] should produce the same results as the method based on the rotation axis and angle. In the method based on quaternions, a closed formula is presented also based on the search for a normed rotation axis and angle in the form of a quaternion. For that, the eigenvector corresponding to the largest eigenvalue of a symmetric 4×4-matrix is computed, which results in the minimizing quaternion. It is important to mention that the solutions by Horn [71; 70] were originally proposed only for the least-squares formulation to register two coordinate systems.

6.4.5 Comparison with physically-based mesh deformation

The interactive simulation of stiff deformable objects is a problem well suited for the application of the high programmability of current graphics hardware. Thus, a hardware-accelerated simulation system for deformable tetrahedral meshes based on implicit integration [149] was developed and is presented here as opposing case to the meshless deformation described above. As in moving least-squares deformations, with the simulation performed on the GPU, rendering the deformed body can be realized directly without the need for readbacks and downloads from/to the graphics hardware. For means of performance comparison, different explicit solvers were implemented on the GPU. This allows us to better compare in terms of performance the GPU implementations of the moving least-squares deformation algorithm and of a traditional physically-based mesh-oriented deformation method.

It is important to mention here, related work that has been developed in the last years. Müller *et al.* [118] presented a approach based on the finite element method for real-time deformations. By estimating the rotational part of the deformation and using linear elasticity, they create plausible animations free of the disturbing artifacts present in linear models and faster than non-linear models. However, since they solve a linear system on the CPU for the implicit integration, its use with large meshes is still limited. Teschner *et al.* [152] perform deformations on low resolution tetrahedral meshes, coupled with high resolution surface meshes used to visualize the deformed body. Explicit Verlet integration on the CPU is used to solve Newton's equation of motion. The actual deformation process is able to handle up to 25000 tetrahedra at interactive rates.

Physically-based simulation on the GPU has been addressed by researchers during the last years to simulate a variety of phenomena [65; 66; 56; 86; 23]. The approach by Georgii *et al.* [56] is of particular relevance in the context of this work. They simulate mass-spring models at interactive frame rates through a GPU-based computation of the Verlet integration method over tetrahedral meshes, where the edges of the tetrahedra represent springs that join the particles. Although the frame rates reported are interactive, instabilities arise for large time steps due to the use of an explicit method and to the stiffness of equations with high spring constants. Thus, here implicit integration on the GPU is addressed.

Implicit integration of the physical model. The physical model is based on the set of N interacting particles in a given tetrahedral mesh. Particle i interacts with the set $\mathcal{N}_i$ of the N_i particles connected to it by an edge. Interaction is represented by a linear spring model, for which the energy function E for two particles i and j is given by

$$E = \frac{1}{2}\kappa_s(\|\mathbf{x}_{ij}\| - L)^2, \tag{6.35}$$

where L is the original distance between the particles, $\mathbf{x}_i$ and $\mathbf{x}_j$ their positions, $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$, and κ_s the spring constant. Given the particles velocities $\mathbf{v}_i$ and $\mathbf{v}_j$, damping forces exerted on particle i from the interaction with particle j are also included. Thus, the forces acting on particle i due to particle j are

$$\begin{aligned} \mathbf{f}_{ij}^{[s]} &= -\frac{\partial E}{\partial \mathbf{x}_i} = \kappa_s(\|\mathbf{x}_{ij}\| - L)\frac{\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|} && (6.36) \\ \mathbf{f}_{ij}^{[d]} &= -\kappa_d(\mathbf{v}_i - \mathbf{v}_j). && (6.37) \end{aligned}$$

The combined force $\mathbf{f}_i$ over particle i is given by $\mathbf{f}_i = \sum_{j \in \mathcal{N}_i}(\mathbf{f}_{ij}^{[s]} + \mathbf{f}_{ij}^{[d]}) + \mathbf{f}_i^{[e]}$, where $\mathbf{f}_i^{[e]}$ is the sum of external forces applied on the particle that do not depend on its position or velocity. Then, the derivatives of the force with respect to the position and the velocity are the matrices given by

$$\begin{aligned} \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j} &= \kappa_s \frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\mathbf{x}_{ij}^T\mathbf{x}_{ij}} + \kappa_s\left(1 - \frac{L}{\|\mathbf{x}_{ij}\|}\right)\left(\mathbf{I} - \frac{\mathbf{x}_{ij}\mathbf{x}_{ij}^T}{\mathbf{x}_{ij}^T\mathbf{x}_{ij}}\right) && (6.38) \\ \frac{\partial \mathbf{f}_i}{\partial \mathbf{v}_j} &= \kappa_d \mathbf{I}. && (6.39) \end{aligned}$$

Arranging the forces, positions and velocities of all N particles in three arrays $\mathcal{F} = [\mathbf{f}_1, .., \mathbf{f}_N]$, $\mathcal{X} = [\mathbf{x}_1, .., \mathbf{x}_N]$ and $\mathcal{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_N]$, respectively, and given the $3N \times 3N$ diagonal matrix $\mathbf{M} = [m_1, m_1, m_1, \cdots, m_N, m_N, m_N]$, where m_i is the mass of particle i, the dynamical problem can be written in terms of the second-order differential equation

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) \qquad (6.40)$$

Given the known position $\mathbf{x}(t)$ and velocity $\mathbf{v}(t)$ of the system at time t, the goal is to find the new position $\mathbf{x}(t+h)$ and the new velocity $\mathbf{v}(t+h)$ at time $t+h$, where h is the time step.

Defining $\mathbf{v} = \dot{\mathbf{x}}$, Equation 6.40 is converted to a first-order differential equation:

$$\frac{d}{dt}\begin{pmatrix} \mathbf{x} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}, \mathbf{v}) \end{pmatrix}. \qquad (6.41)$$

Letting $\Delta\mathbf{x} = \mathbf{x}(t+h) - \mathbf{x}(t)$ and $\Delta\mathbf{v} = \mathbf{v}(t+h) - \mathbf{v}(t)$, the implicit Euler method approximates $\Delta\mathbf{x}$ and $\Delta\mathbf{v}$ as

$$\begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{v} \end{pmatrix} = h\begin{pmatrix} \mathbf{v}(t) + \Delta\mathbf{v} \\ \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}(t) + \Delta\mathbf{x}, \mathbf{v}(t) + \Delta\mathbf{v}) \end{pmatrix} \qquad (6.42)$$

Following the groundbreaking work by Baraff and Witkin [16], $\Delta\mathbf{x} = h(\mathbf{v}(t) + \Delta\mathbf{v})$ in the lower part of Equation 6.42 is replaced and the first order approximation of a Taylor series expansion on $\mathbf{f}$ is taken, to form the system $\mathbf{A}\Delta v = \mathbf{b}$

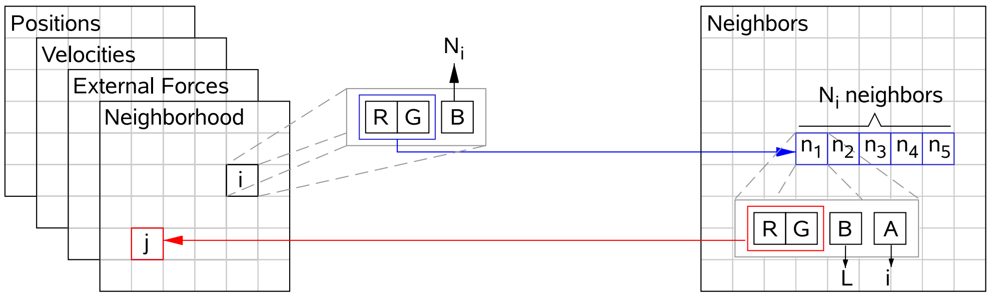


Figure 6.13: Textures `neighbors` and `neighborhood` hold the information of the neighboring particles.

$$\left(\mathbf{I} - h\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)\Delta\mathbf{v} = h\mathbf{M}^{-1}\left(\mathbf{f}(t) + h\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v}(t)\right) \tag{6.43}$$

which must be solved for $\Delta\mathbf{v}$ in order to find $\mathbf{x}(t+h)$ and $\mathbf{v}(t+h)$.

Hardware-accelerated simulation. To solve the linear system of Equation 6.43 on the GPU, the input data is stored in 32 bit floating point textures. These textures hold the state information (vectors $\mathbf{x}$ and $\mathbf{v}$), the externals forces $\mathbf{f}^{[e]}$, connectivity information, and partial results obtained during the simulation.

The input data is stored in five 2D textures, namely, `external_forces`, `positions`, `velocities`, `neighbors`, and `neighborhood`, with dimensions given by $\lceil\sqrt{N}\rceil$, with the exception of `neighbors` whose dimensions are $\lceil\sqrt{\sum_{i=1}^{n} N_i}\rceil$[1].

For each particle i, texture `positions` holds its position in space, while texture `velocities` holds its velocity and mass. The sum of the external forces is stored in `external_forces`. Connectivity information is stored in two separate textures (See Figure 6.13). Texture `neighborhood` stores a pointer to the position of texture `neighbors`, where the information of the N_i neighbors of the particle is stored. This information includes a pointer back to the position of the neighbor in the first four textures, the original distance between the particle and the neighbor, and the index of the particle. The number of neighbors N_i is stored in one channel of texture `neighborhood`. Although κ_s, κ_d, and h are passed as environment parameters, κ_s could be stored alternatively in the remaining channel of texture `neighborhood` to allow the use of different local material properties.

This arrangement maps nicely to hold the sparse non-banded matrix computed when forming the linear system. Each row of N_i neighbors in texture `neighbors` can be regarded as being the non-zero non-diagonal positions of

[1]Better choices for the dimensions of these textures could be achieved using the results given in the work by Bolz [23].

the i-th row in the matrix. This holds, since only the elements of the matrix corresponding to two connected particles are non-zero.

A typical implementation of the simulation algorithm is given in the following, where the description of the GPU implementation of each step of the algorithm is detailed. The first step computes $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$, $\frac{\partial \mathbf{f}}{\partial \mathbf{v}}$ and $\mathbf{f} = \mathbf{f}^{[s]} + \mathbf{f}^{[d]} + \mathbf{f}^{[e]}$. Two rendering passes are performed to compute the derivatives and forces. In the first pass, the non-diagonal elements of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ are calculated. For this, a quadrilateral covering a viewport of size $\lceil\sqrt{\sum_{i=1}^{N} N_i}\rceil$ is rendered to generate the fragments representing the non-zero non-diagonal elements of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$. Since each element of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is a 3×3-matrix, the result of each fragment has 9 elements. Thus, the results are rendered to three target textures `non_diagonal_dfdx`k, $k = 0, 1, 2$. The texture `non_diagonal_dfdx`k will hold the results of the non-zero non-diagonal elements of the $(3 \times i + k)$-th row of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$, in the texels corresponding to the neighbors of the i-th particle. Textures `positions`, `neighbors`, and `neighborhood`, as well as the parameters κ_s and h, are input to this pass.

In the second pass the diagonal elements of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ and the force vector f are computed. Each diagonal element of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is given by the negation of the sum of the non-diagonal elements in its row. Thus, in the second rendering pass, the diagonal elements are computed by generating $(\lceil\sqrt{N}\rceil)^2$ fragments, and summing in each fragment the results of the previous rendering pass corresponding to its neighbors. Therefore, textures `non_diagonal_dfdx`k are inputs to the current rendering pass. To access the information in textures `non_diagonal_dfdx`k, the textures `neighborhood` and `neighbors` are required. As the combined forces are also computed in this pass, textures `positions`, `velocities`, and `external_forces` are also needed as input. Thus, fragment i calculates the 3×3-matrix corresponding to the i-th diagonal element of $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$, and the combined force corresponding to particle i. The result is stored in four target textures: `diagonal_dfdx`k; $k = 0, 1, 2$ and `forces`.

The third step is forming the linear system $\mathbf{A}\Delta\mathbf{v} = \mathbf{b}$ given by Equation 6.43. The linear system is formed in three rendering passes. In the first pass the right side $\mathbf{b}$ of Equation 6.43 is calculated by generating $(\lceil\sqrt{N}\rceil)^2$ fragments. Each fragment calculates three elements of vector $\mathbf{b}$ to be stored in texture `b`. Since matrix $\mathbf{M}^{-1}$ is diagonal, it is only needed to loop over the neighbors of the particle i corresponding to the fragment to calculate $\mathbf{f}(t) + h\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\mathbf{v}(t)$ and multiply the result by $\frac{h}{m_i}$. For this, textures `non_diagonal_dfdx`k, `diagonal_dfdx`k, and `forces` are required, so as the textures `neighbors`, `neighborhood`, and `velocities` and the parameter h.

Next, the matrix $\mathbf{A}$ on the left side of the linear system must be computed. In the second pass, the non-diagonal elements given by $-h\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - h^2\mathbf{M}^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$ are

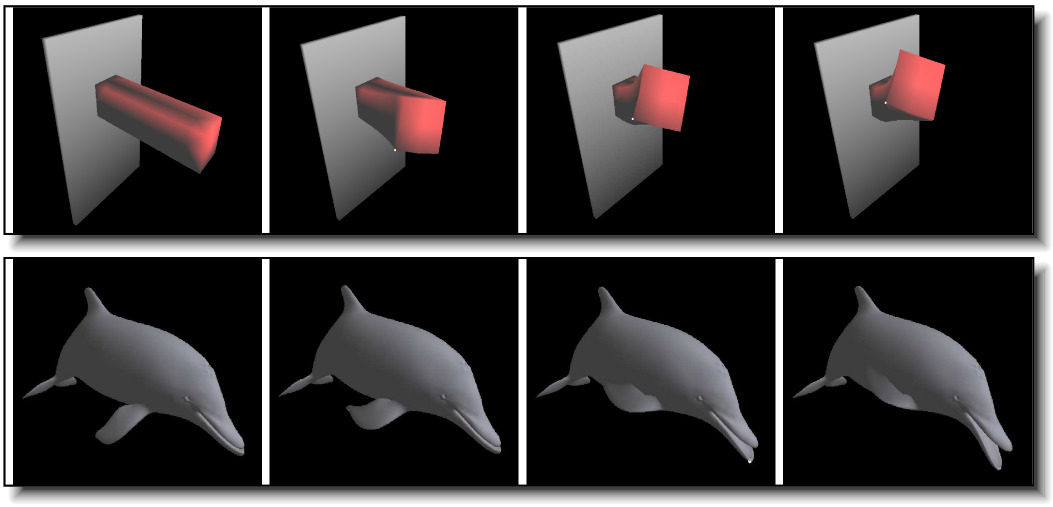


Figure 6.14: Surface rendering of deformed tetrahedral meshes.

calculated. To that end, we generate $(\lceil\sqrt{\sum_{i=1}^{N} N_i}\rceil)^2$ fragments. Each fragment multiplies the input from `non_diagonal_dfdx`k by $-\frac{h^2}{m_i}$, to obtain a partial result. Note that the index i of the corresponding particle (row) is needed, which can be fetched from the alpha channel of texture `neighbors`. Then, $-\frac{\kappa_d h}{m_i}$ is added to the diagonal elements of the partial result, and this final result is written to the textures `non_diagonal_A`k, $k = 0, 1, 2$.

To compute the diagonal elements, $(\lceil\sqrt{n}\rceil)^2$ fragments are generated, providing as input textures `diagonal_dfdx`k, and their content is multiplied by $-\frac{h^2}{m_i}$ for fragment i. $1 - \frac{\kappa_d h N_i}{m_i}$ is added to the diagonal elements and then the result is written to textures `diagonal_A`k; $k = 0, 1, 2$. Note that textures `velocities` and `neighborhood` are also needed as input due to m_i and N_i.

The third and final pass solves the linear system for $\Delta\mathbf{v}$ and updates the vectors $\mathbf{x}$ and $\mathbf{v}$. With the textures `diagonal_A`k, `non_diagonal_A`k, and `b` holding the matrix $\mathbf{A}$ and vector $\mathbf{b}$, the problem fits nicely to the GPU-based implementation of the Conjugate Gradients algorithm proposed by Krüger *et al.* [86]. The arrangement of the data we use differs from the one proposed by Krüger and is more similar to the one proposed by Bolz *et al.* [23]. Thus, the GPU matrix operations proposed by Krüger were fitted to work with our arrangement. One of the major differences with both works is the implementation of the reduction operator. Instead, a two pass reduction operator was implemented, compared to the $\log N$ passes needed by Krüger and Bolz. In the first pass, a reduction by a factor of 255 is performed. This result is then combined in a second pass, so it is possible to handle vectors of up to 255^2 elements. Further passes or nested `LOOP` instructions would be needed for larger vectors. We also eliminate the need for multiple passes

	Mesh size	Frame rates [fps]				CPU Euler [fps]	
	Tetrahedra	Expl. Euler	Verlet	Veloc. Verlet	Impl. Euler	Explicit	Implicit
Bar	80	3880	3408	3454	282	5824	67
Dolphin	13766	1039	764	764	113	102	*
Panda	17312	945	612	614	100	80	*
Elephant	24106	900	539	541	48	58	*
Knee	112299	169	119	119	22	14	*
Foot	156612	220	149	149	12	10	*

* Maximum response time exceeded.

Table 6.3: Performance in fps of the integration methods on the GPU and CPU.

looping on the non-zero elements of each row and using the information stored on the alpha channel of texture `neighbors` to access the corresponding element in the vector to be multiplied by the matrix. Each fragment generated performs this operation and then adds the result to the contribution of the diagonal element.

Once the linear system is solved, the solution vector $\Delta\mathbf{v}$ is used as input to a final rendering pass, where n fragments are generated to update the position and velocity of each particle. The result is rendered to textures `positions` and `velocities`, and then the next iteration of the simulation is started. During rendering, to access the deformed vertex positions, each vertex fetches its correct position from the resulting textures, which is then transformed and passed to the rasterizer. This allows to avoid costly readback operations from the GPU.

Figures 6.14 and 6.15 show deformations performed on two tetrahedral meshes using our approach. Results describing the performance of the algorithms are presented here. All performance measurements were carried out on a standard PC equipped with an NVIDIA GeForce 7800 graphics board, a 3.8 GHz P4 CPU, and 2GB RAM. Table 6.3 shows the comparative results of the GPU-based implicit Euler and the straightforward GPU implementations of the explicit Euler, Verlet and velocity Verlet solvers, which are thus not described here. Frame rates for the integration process (not including rendering time) are given in fps. Timings for the naive CPU versions of the implicit and explicit Euler are also included. In some cases, the CPU implementation of the implicit Euler was not able to solve the equation before the program aborted due to exceeded time without response. In debugging mode, the program solved the linear system successfully with a significant increase of the processing time. It is important to remark that, for the Bar mesh, it was possible to use a maximum time step of $h = 0.001s$ for $\kappa_s = 3000$ with the explicit integration methods. With higher κ_s or larger h, instability occurred. On the other hand, with the implicit Euler and $h = 0.01s$ no stability

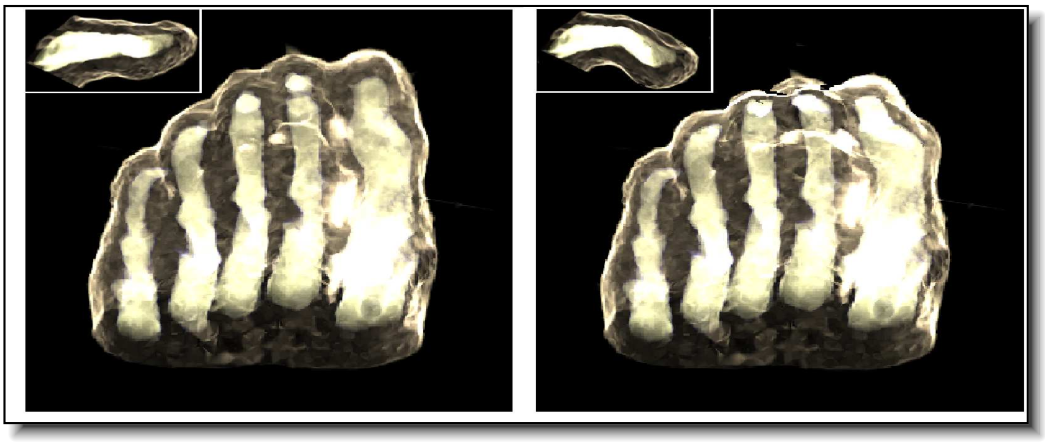


Figure 6.15: Original and deformed mesh of the Foot dataset. A detail on a toe is shown in the upper left corner of each image.

problem was found using κ_s larger than 30000.

In terms of performance, physically-based deformations solved on the GPU with explicit methods are as fast as computing a low-sampled displacement map ($64 \times 64 \times 64$), for rigid deformations, on the GPU for a similar number of elements (cells). However, as mentioned before, explicit methods are in general instable. When compared to the GPU implementation of the implicit Euler method, the computation of the displacement map performs faster. Nonetheless, both approaches provide interactive frame rates. As initially predicted, physically-based deformations and meshless deformations perform better in different tasks. While physically-based deformations are suitable for simulation tasks and visualization problems where realism is important, it has been observed that moving least-squares deformations are more suitable for exploratory tasks due to the ease of manipulation, for instance, in illustrative, educational and entertainment software. One task that is difficult to accomplish with both physically-based and non-physically-based deformations is performing cuts. Correa *et al.* [36] addressed this issue for the case of displacement maps by introducing discontinuous displacements. In the case of moving least-squares deformations this is easily accomplished by modifying the weighting function as was shown above. Finally, meshless physically-based deformation has been addressed by other authors and the interested reader can find a number of works on the field, *e.g.,* [119; 80; 120; 125].

CHAPTER

7 MESHLESS METHODS IN VISUALIZATION

In this work, surface and volume data reconstruction and rendering were addressed with meshless techniques. Most of the techniques presented are based on the moving least-squares approximation method, which gained popularity in the last years within the computer graphics community, specially for surface approximation from unorganized point sets. The meshless methods developed by the surface reconstruction community generate astonishingly good results compared to traditional methods based on combinatorial structures, which can be regarded as mesh-based methods and dominated the area for decades.

However, despite the success of meshless surface reconstruction methods, some problems persist and new problems arose. Thus, the approaches developed in the context of the work reported in this thesis, initially targeted problems found in meshless surface reconstruction algorithms. The resulting set of techniques can be seen in Figure 7.1. The first problem addressed resulted in the curvature-driven projection operator proposed in Section 4.2, which was developed to reduce the number of operations needed for calculating the polynomial local approximations to the surface. This was done, since the number of local approximations that must be calculated to fully represent or render a surface using moving least-squares is very large. With the projection operator based on curvature it was possible to model the local approximations using non-complete quadratic polynomials. This can be exploited in different scenarios. For instance, graphics hardware implementations can benefit from the reduced size of the matrix of the normal equation, since 4×4-matrices are natively supported by current commodity graphics cards. With this goal in mind, a graphics hardware implementation of the original moving least-squares surfaces definition was presented. This made it possible to obtain interactive framerates for rendering a surface. One further contribution of the work on the curvature-driven projection operator was the computation of the curvature itself for unorganized point sets. As shown in the literature, the curvature information can be used not only to have a quantitative measure to analyse the surface, but for rendering purposes, for instance, for performing non-photorealistic rendering or accessibility shading.

A number of works on implicit moving least-squares surfaces was developed the last years by many authors, addressing sampling, reconstruction and performance issues. However, one of the major drawbacks of the techniques presented is the linear nature of the surface approximation. Thus, recent results on approxi-

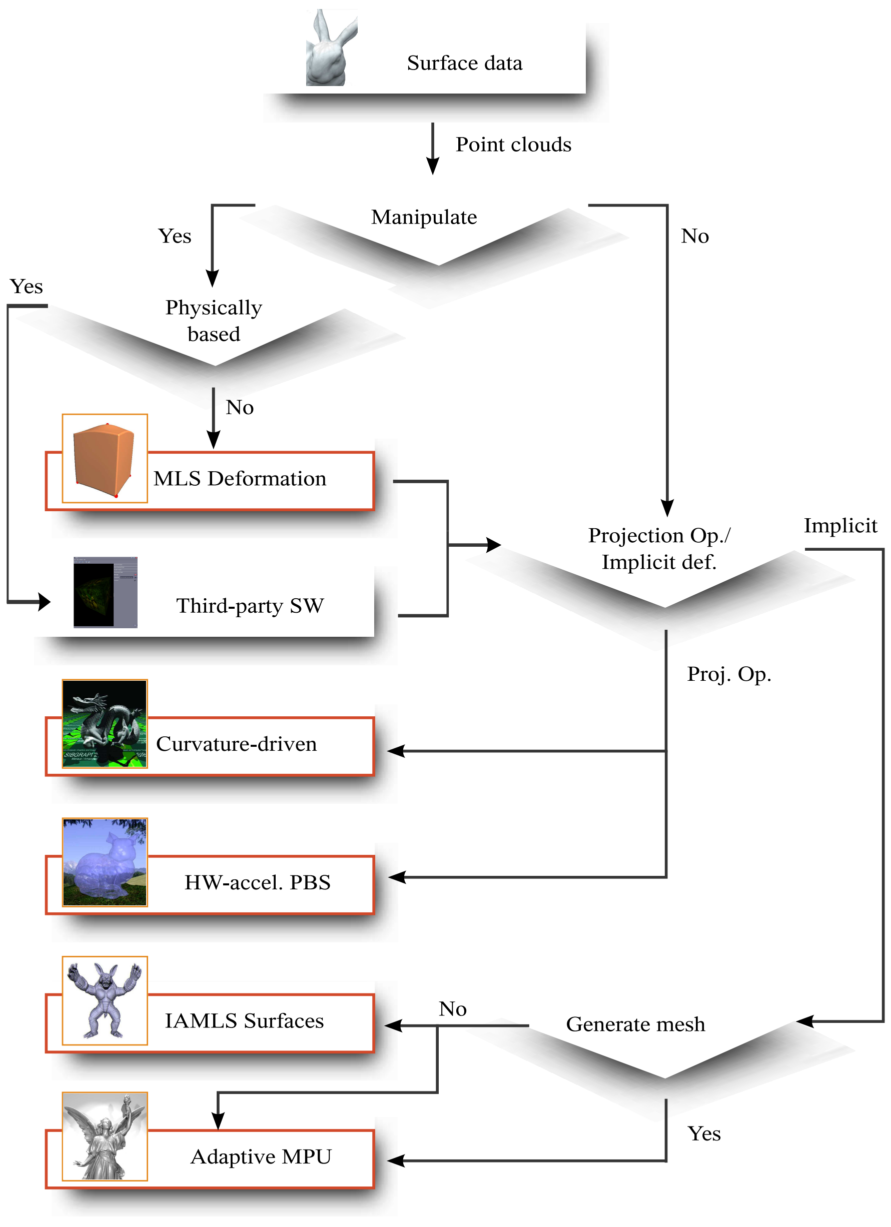


Figure 7.1: Guideline for the use of a (incomplete) set of meshless techniques in the context of surface approximation. The methods proposed in this thesis are marked with red borders (see color plates).

mate approximation were used in this work to combine the simplicity of implicit definitions with the high-order approximation of projection operators. The resulting method is presented in Section 4.3. This method benefits from the matrix-free nature of approximate approximations, which allows the computation of higher-order approximations without having to solve systems of equations. This allowed the definition of an implicit function, whose zero set provides an approximation to the surface that, differently from previous definitions, is not linear. Also, it was shown that this simplicity can be exploited to introduce new mechanisms to model specific features of the surface, specifically, sharp edges. Bilateral filtering was introduced into the surface definition as an iterative processing to model sharp features in the model. Despite the results being an approximation to the moving least-squares approximation, the fact that no system of equations must be solved helps to avoid numerical instabilities.

Another family of meshless surface approximation methods is comprised by the partition of unity implicts. These implicit definitions suffer from robustness and algorithmic problems. The latter refer specifically to the need for different data structures to perform the surface approximation and to extract the output mesh. Also, spatial adaptiveness is sometimes not enough to accommodate complex surfaces. Thus, a method based on orthogonal polynomials and on the J^1_A data structure was devised (Section 4.4), which provides not only spatial adaptiveness but also approximation order adaptiveness. Specifically, the degree of the polynomial approximation matches the local complexity of the surface. However, with this approach the robustness of the method worsens due to the oscillations produced by high-degree polynomials. To attack this problem, different algorithmic solutions were combined with satisfactory results.

The potential advantages of using meshless approximation methods for volume data visualization were also explored (Figure 7.2). Meshless reconstruction methods in this context were relegated to scattered data in the past and were basically built upon partition of unity, inverse distance weighting and similar approaches. The work in this direction was started by directly extending the moving least-squares surface definitions to isosurfaces and surfaces located at regions of high gradient magnitude. A combination, in the sense of a predictor-corrector approach, of weighted least-squares and moving least-squares was defined, which increased the domain of the definition while maintaining the quality of the representation. The result of this technique is a smooth two-manifold representing some feature in the volume.

The flexibility of points was used in several works to render surfaces. This flexibility was the reason why rendering dynamic surfaces extracted from volumetric data using points was explored in Section 5.3. Specifically, the interactive rendering of streamsurfaces, and the novel path-surfaces, was addressed. Commodity graphics hardware capabilities were used to generate a dense sampling of the streamsurface or path-surface. This sampling was then used to render a closed surface with splatting. Interactive frame rates were achieved, allowing the user to place the probe for generating the streamsurface or path-surface and to animate the path surface in real time.

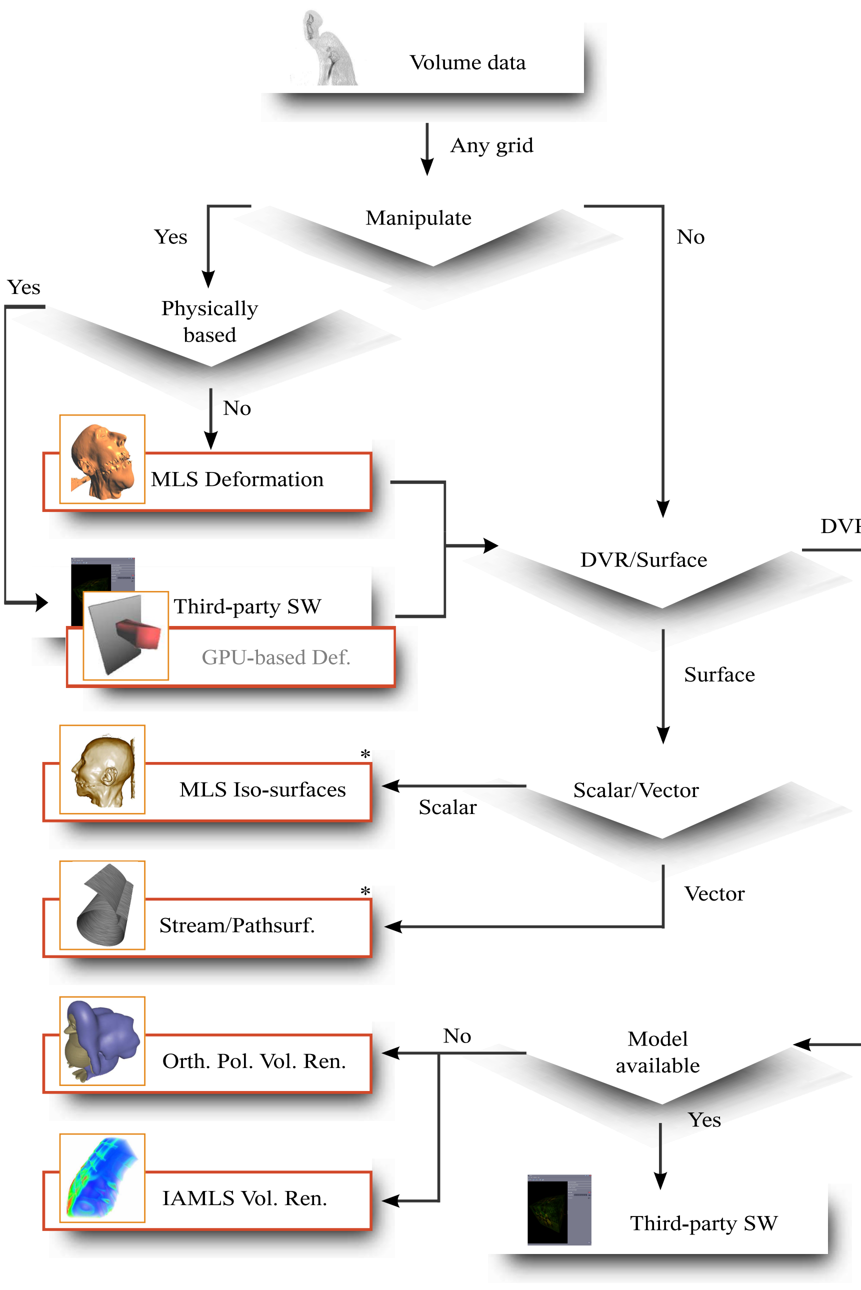


Figure 7.2: Guideline for the use of a (incomplete) set of meshless techniques in the context of volume visualization. The methods proposed in this thesis are marked with red borders (see color plates).

Methods for rendering volumetric data stored in a wide range of mesh types were developed as well (Sections 6.2 and 6.3). The goal was to exploit the flexibility of meshless approximation methods for defining a reconstruction technique independent of the mesh type. The first approach to solve this problem led to the combined use of detail-preserving weighting functions and moving least-squares. While the reconstruction results were promising, numerical instabilities often occurred and a large number of systems of equations had to be solved. Thus, orthogonal polynomials were used to tackle both problems with good results. Performance increased considerably while improving stability enough for the purposes of visualization. Despite this fact, a graphics hardware implementation proved to be still slow for visualization tasks. Therefore, approximate approximation was used to increase performance. Although the method based on orthogonal polynomials produces more accurate solutions than the method based on approximate approximation, the latter proved easy to implement and fast to compute compared to the former.

Implementing orthogonal polynomials on the graphics hardware for other purposes might deliver interactive frame rates. This is the case for the non-linear polynomial moving least-squares deformations proposed in Section 6.4. This method is based on recent work on moving least-squares deformations in three dimensions, which provided closed solutions for computing affine, similarity and rigid deformations. Thus, in some sense, the non-linear polynomial deformations proposed complete this set of moving least-squares deformations since by using orthogonal polynomials a closed solution for higher-order polynomial approximations (interpolations in this case) can be obtained. This fact was exploited, as hinted before, to develop hardware-accelerated algorithms for the complete set of moving least-squares deformations, which allowed to achieve interactive frame rates. Although these results were used for volume deformation, surface deformation can be as well addressed with the same concept as hinted in Figure 7.1. It is maybe worth mentioning that a hardware-implementation of a physically-based algorithm for deforming tetrahedral meshes was also developed and is included within the discussion of meshless volume deformation as a comparative model.

Albeit being conceptually interesting and setting a base for future work, the meshless methods presented in this thesis need to be accelerated to deliver interactive frame rates. Although it is highly probable that next generation of graphics hardware will be able to accomplish such task, the size of the datasets to be visualized will also grow. However, we believe that the use of new results from approximation theory can help overcome this problem. Also, rigorous mathematical proofs of convergence must be developed to better sustain the methods. Nonetheless, in the same way that meshless techniques have proved to be advantageous in comparison to mesh-based techniques for surface reconstruction, not only from unorganized point sets but also from polygon soups and polygonal meshes, they could also deliver interesting results in the context of volume data visualization.

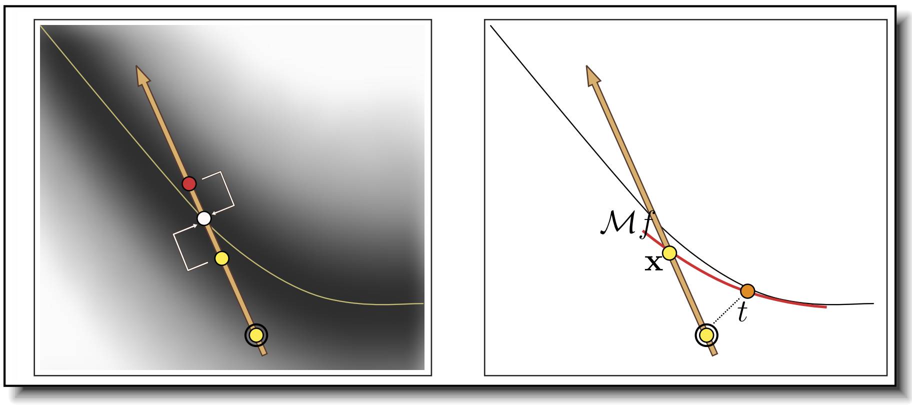


Figure C.1: Ray-casting implicit surfaces (left) and surfaces defined as the set of static points of some projection operator (right). For implicit definitions, a surface cross is found by sampling the ray at regular intervals. With the two last sampled points, the bisection method is applied to find an approximation to the intersection point. In the case of projection operators, the approximate intersection is projected onto the surface. If the distance t to the projection is less than a threshold, the current approximation is the intersection point. Otherwise a local approximation $\mathcal{M}f$ to the surface is computed and its intersection with the ray defines the new approximate intersection $\mathbf{x}$. In both cases the process starts with a point near the surface ∂S depicted by the circled point.

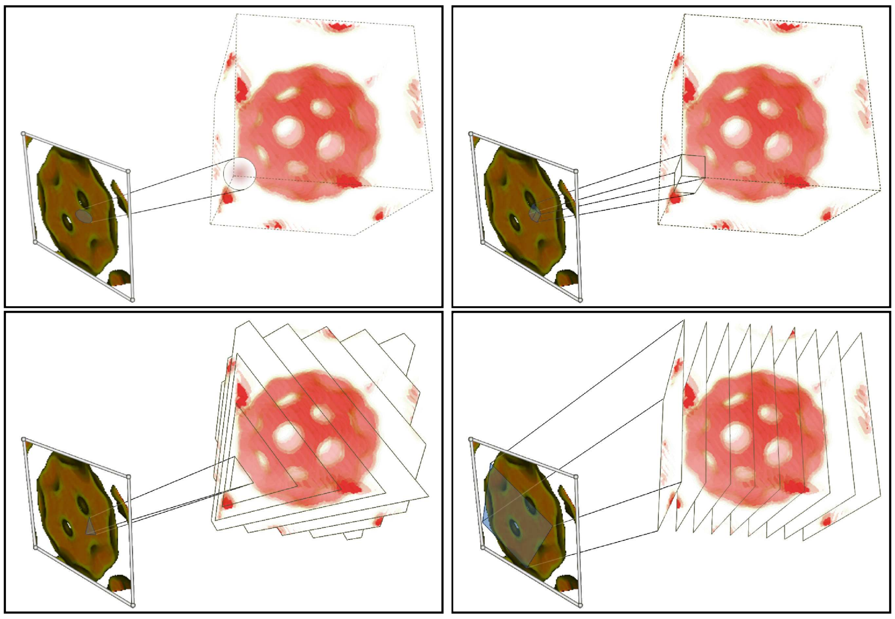


Figure C.2: Object-order volume rendering algorithms. Clockwise (from top left): splatting, cell projection, object-aligned, and view-aligned texture-based volume rendering. Note that splatting and cell projection are not restricted to regular grids.

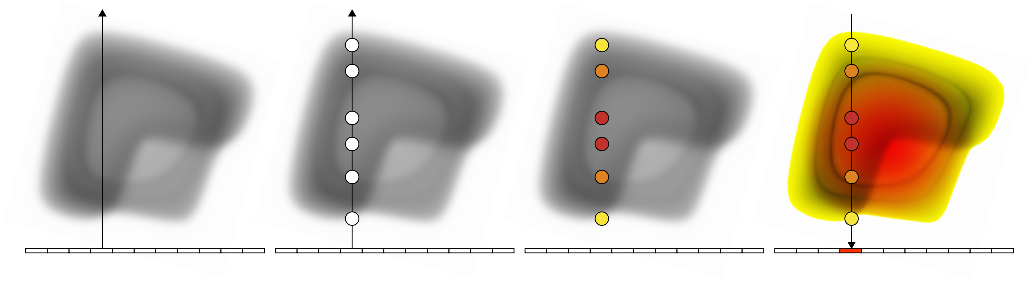


Figure C.3: Volume ray-casting. From left to right: at least one ray is traced for each pixel in the image (ray casting), on each ray the volume is sampled a number of times (sampling), the contribution of each segment is computed (shading), and the contributions of all ray segments are composited to determine to final color of the pixel (composition).

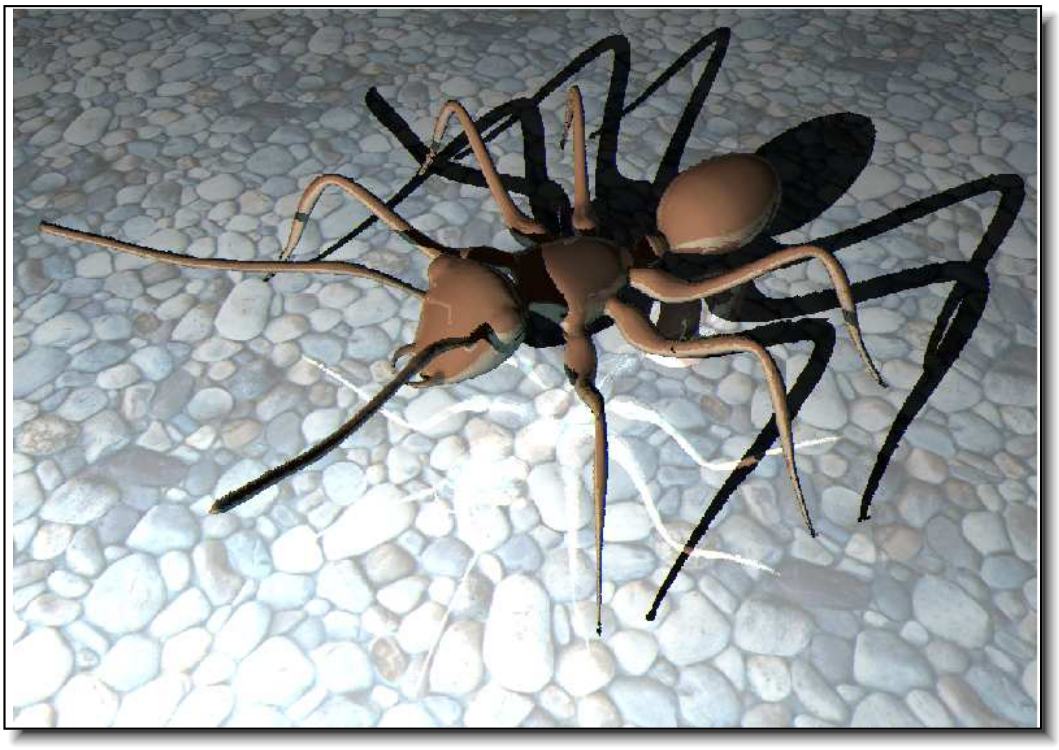


Figure C.4: Rendering of the approximate surface for the EtiAnt dataset obtained with the curvature-driven projection operator described in this section.

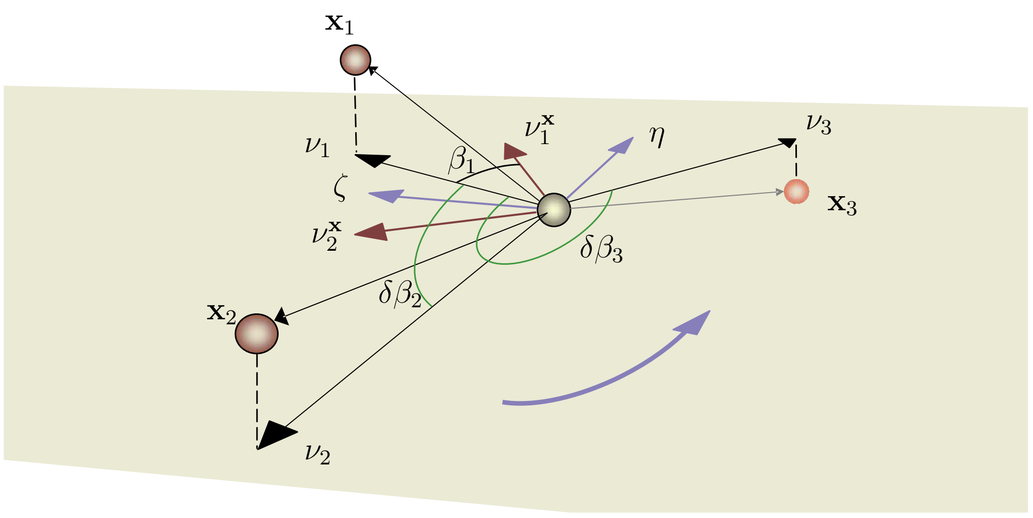


Figure C.5: Estimating directional curvatures on the approximated tangent plane at $\mathbf{x}$.

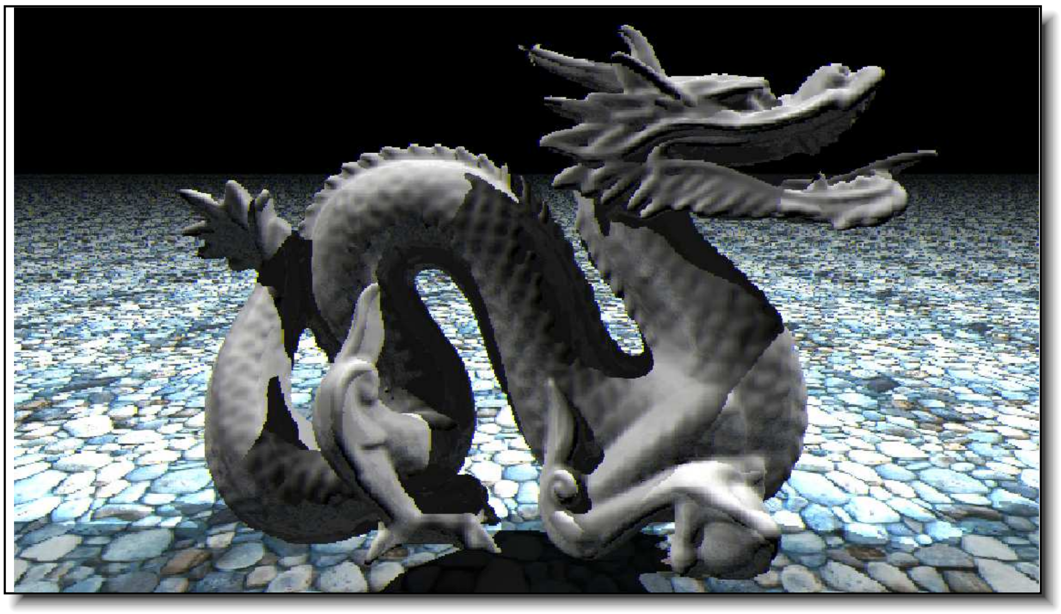


Figure C.6: Approximated surface for the Stanford Dragon obtained as the zero set of the implicit function based on approximate moving least-squares approximation.

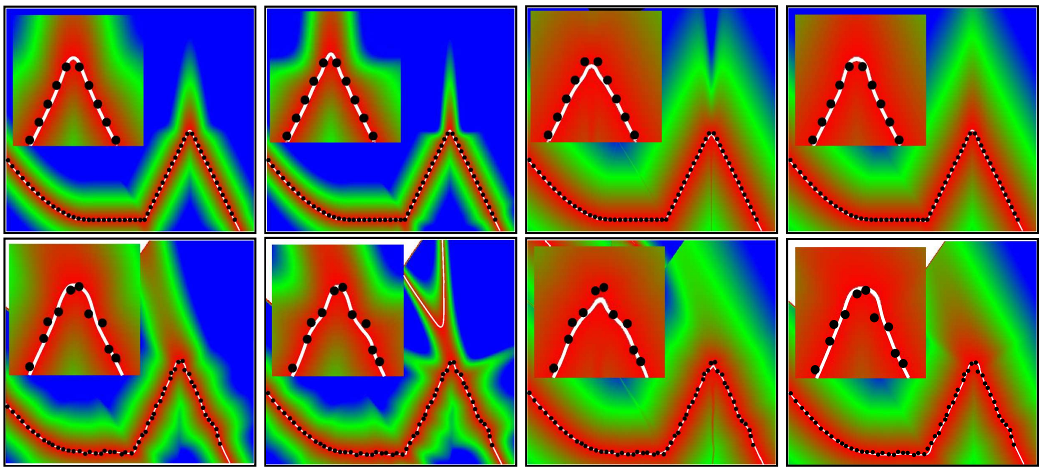


Figure C.7: Plot of the value of the implicit function for a regularly (top) and an irregularly (bottom) sampled dataset. From left to right: AMLS implicits with 5 iterations, AMLS implicits with 20 iterations, Adamson and Alexa's implicits and Kolluri's implicits. The white line shows the zero set of the function while colors map the value of the implicit function with red corresponding to low values and blue to high values.

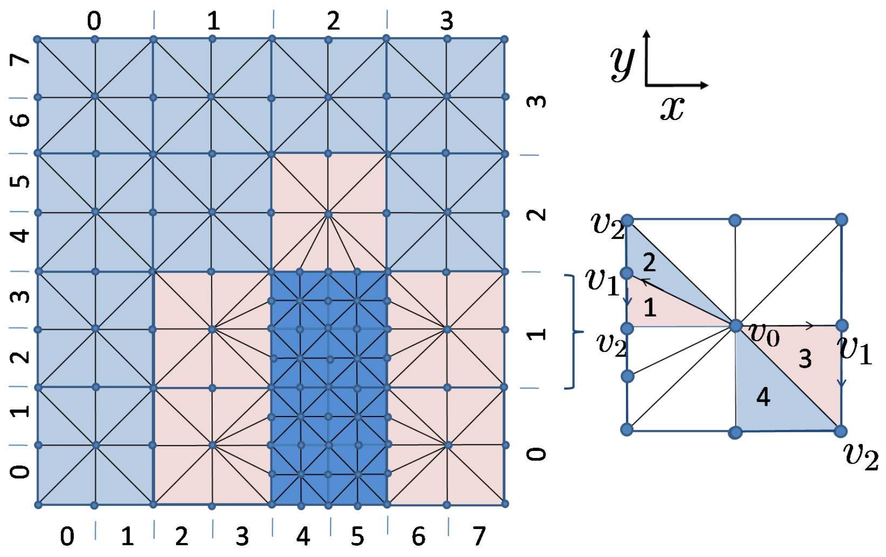


Figure C.8: The J^1_A triangulation: on the left, a sample two-dimensional adaptive triangulation and, on the right, examples of pivoting operations.

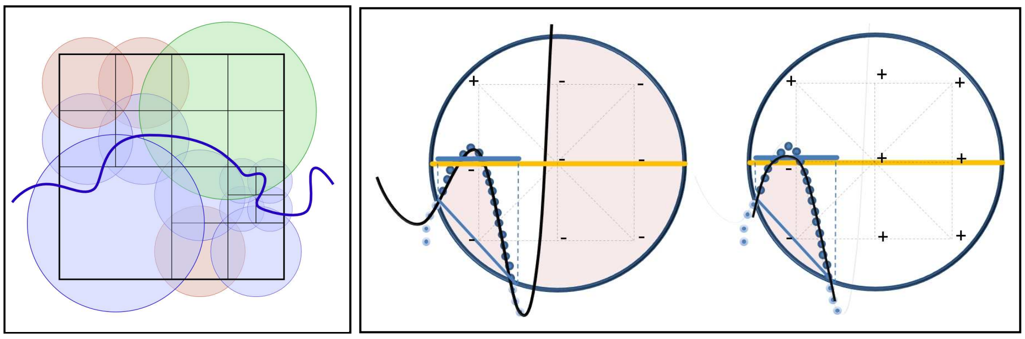


Figure C.9: The figure in the left side depicts the behavior of the J_A^1 during function approximation. The figure in the right side shows an illustration of the effect of the coverage domain on the polynomial approximation. The left side of this figure depicts a case that can arise when a high-order polynomial is used to approximate the surface inside the local domain (blue circle). Since a large region of this domain is void of points, the polynomial approximation may oscillate. Thus, the coverage domain (blue line) is computed and the ratio of the area of the coverage domain and the area of the plane (yellow line) is calculated. This ratio determines the degree of the polynomial used. Decreasing the degree of the polynomial when this ratio is below a threshold reduces the oscillation as can be seen on the right side of the figure.

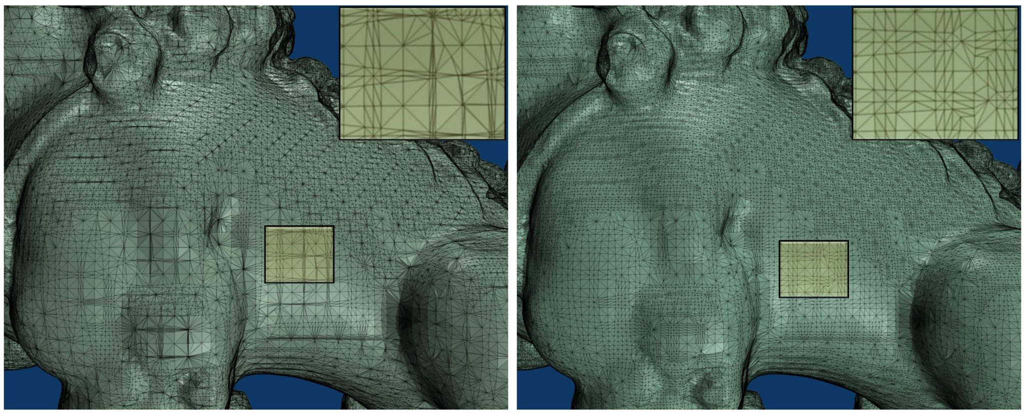


Figure C.10: Comparing the iso-mesh produced from J_A^1 (left) against the iso-mesh obtained from J_A^1 with displacement (right).

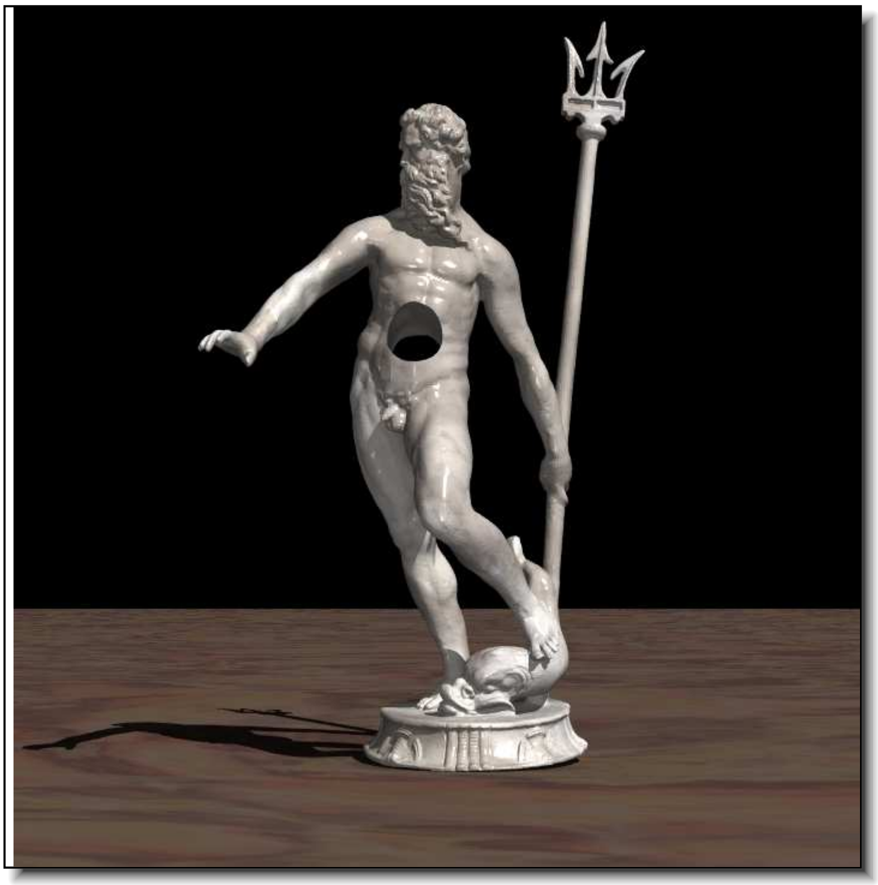


Figure C.11: A CSG difference operation involving the Neptune model and a cylinder.

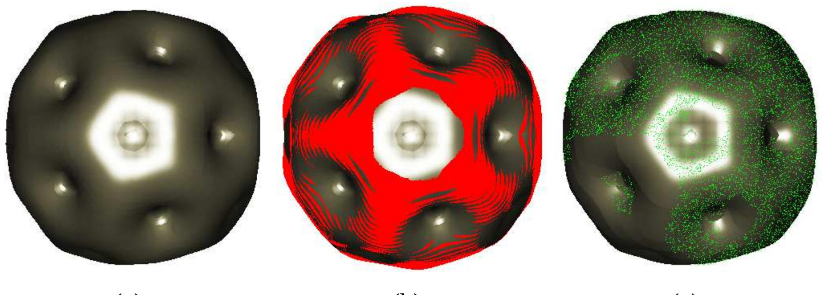


Figure C.12: The Bucky Ball dataset. (a) The final result of applying the predictor-corrector method. (b) The points projected by the predictor at a distance greater than a pre-defined threshold are shown in red. (c) The output points from the predictor projected by the corrector at a distance greater than the threshold are shown in green.

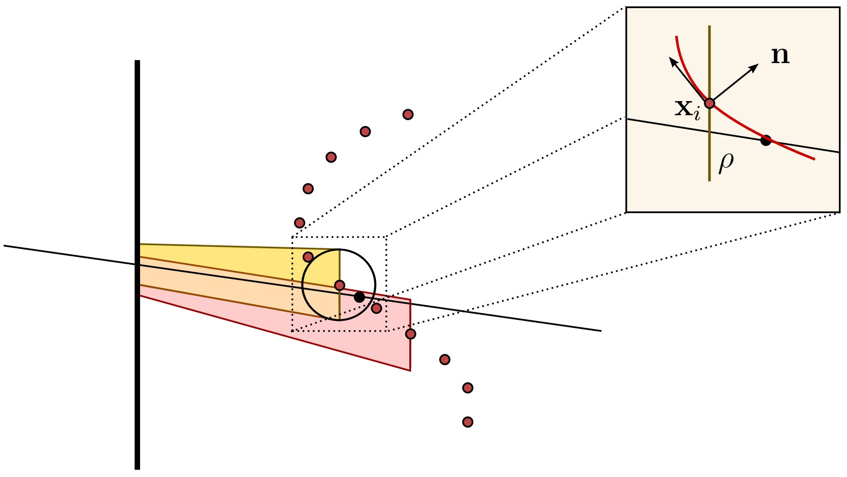


Figure C.13: Calculating the intersection of the ray with the local approximation stored in each sample point.

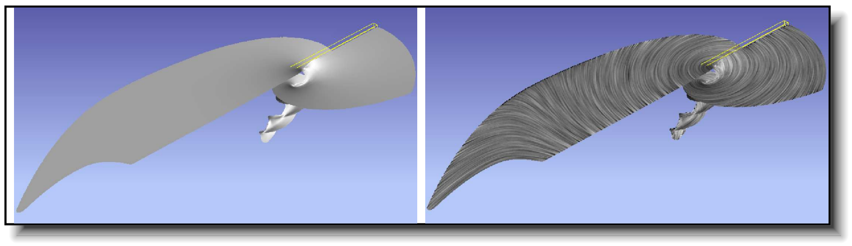


Figure C.14: Visualization of the flow field of a tornado with: (left) a point-based stream surface; (right) the combination of a stream surface and texture-based flow visualization to show the vector field within the surface. Each stream surface is seeded along a straight line in the center of the respective image.

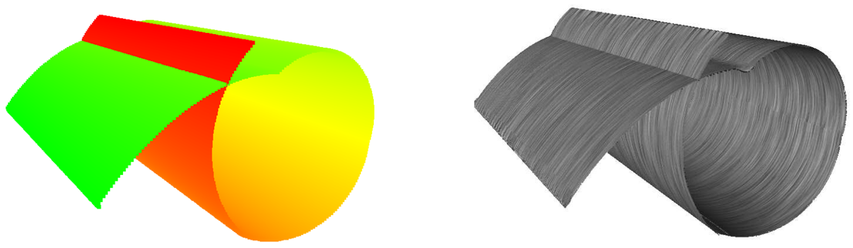


Figure C.15: Path surface of an unsteady flow: on the left side, the time of the unsteady flow field is shown by colors (red for for early times, green for later times); on the right side the combination of the path surface and time-dependent LIC is illustrated.

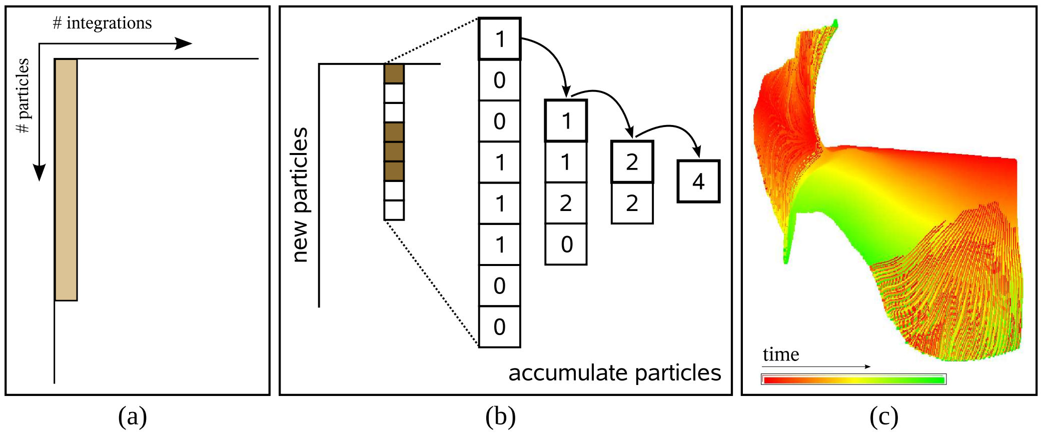


Figure C.16: Illustration of different steps of the algorithm: (a) during the initialization of the particles texture only one column is rendered (the height of the strip represents the number of initial particles) and (b) during creation of the binary tree the new particles build the highest level and the contents are summed up until the root contains the overall number of particles to be inserted. In (c) the lifetime of the individual particles is shown. The color gradient is defined from red (at $t = 0$) to green and illustrates the increasing lifetime. The areas with red lines at the left and bottom-right parts of the image show regions with many new streamlines.

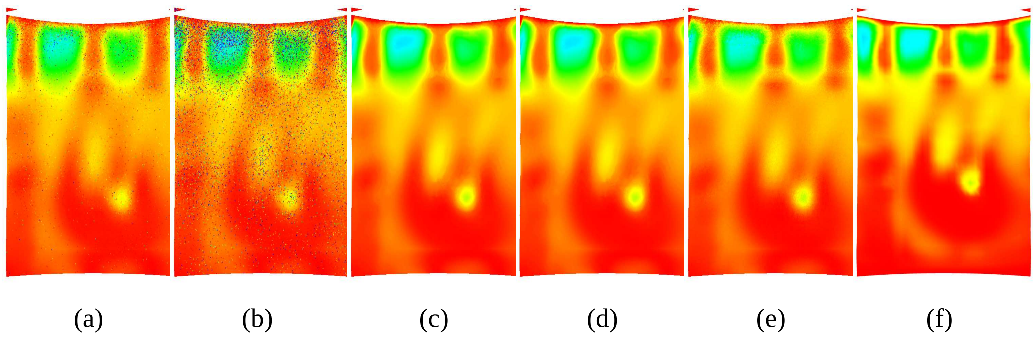


Figure C.17: Volumetric data approximation for a slice of the Combustion Chamber dataset: (a) Gauss-Jordan with pivoting, (b) Conjugate Gradient on normal equations, (c) QR, (d) SVD and (e) orthogonal polynomials. Noise represents evaluation points where instabilities led to a poor approximation. In (f), the result with linear interpolation on the original mesh is shown. Note that, for these results, only `float` precision was used to increase the numerical instability. In practice, choosing a sufficiently large support and using `double` precision, orthogonal polynomials produce results visually indistinguishable from those obtained with QR and SVD.

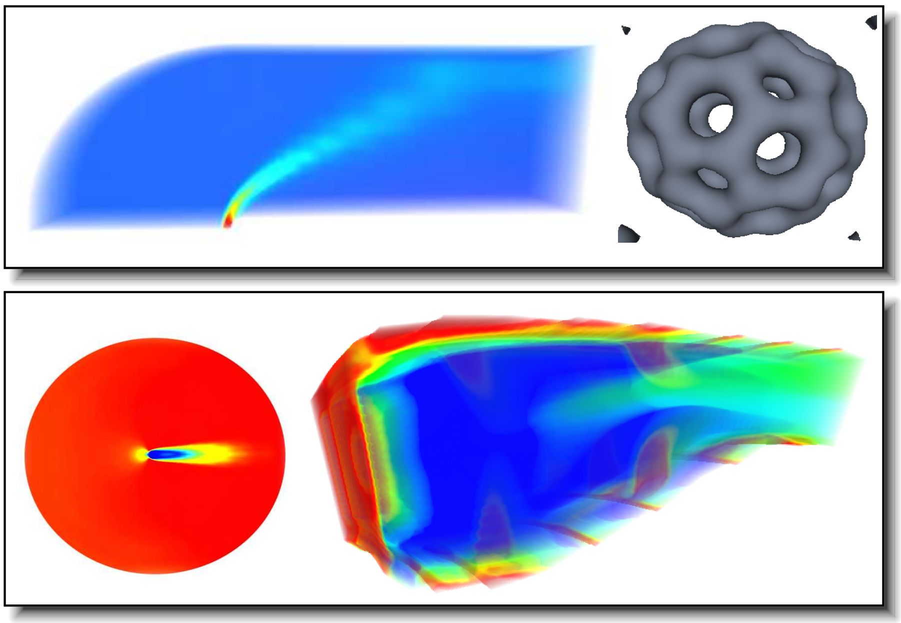


Figure C.18: Renderings of the Blunt Fin, Bucky Ball, Oxygen Post, and Combustion Chamber datasets.

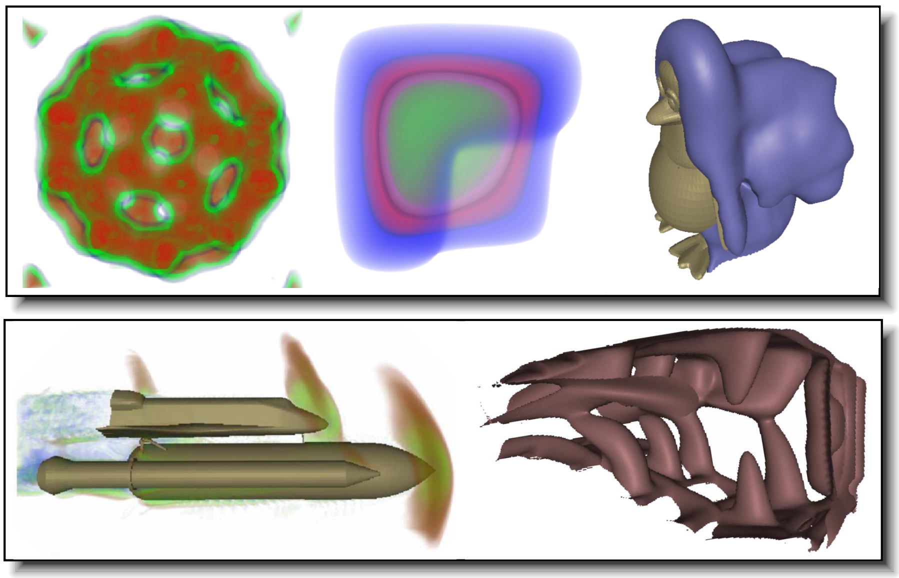


Figure C.19: Volume and isosurface rendering of different data. From left to right: the Bucky Ball structured dataset, the Heat Sink unstructured dataset, the Penguin adaptive-mesh-refinement mesh, the Space Shuttle Launch Vehicle multiblock dataset with multiple overlapping curvilinear meshes and the Combustion Chamber curvilinear dataset.

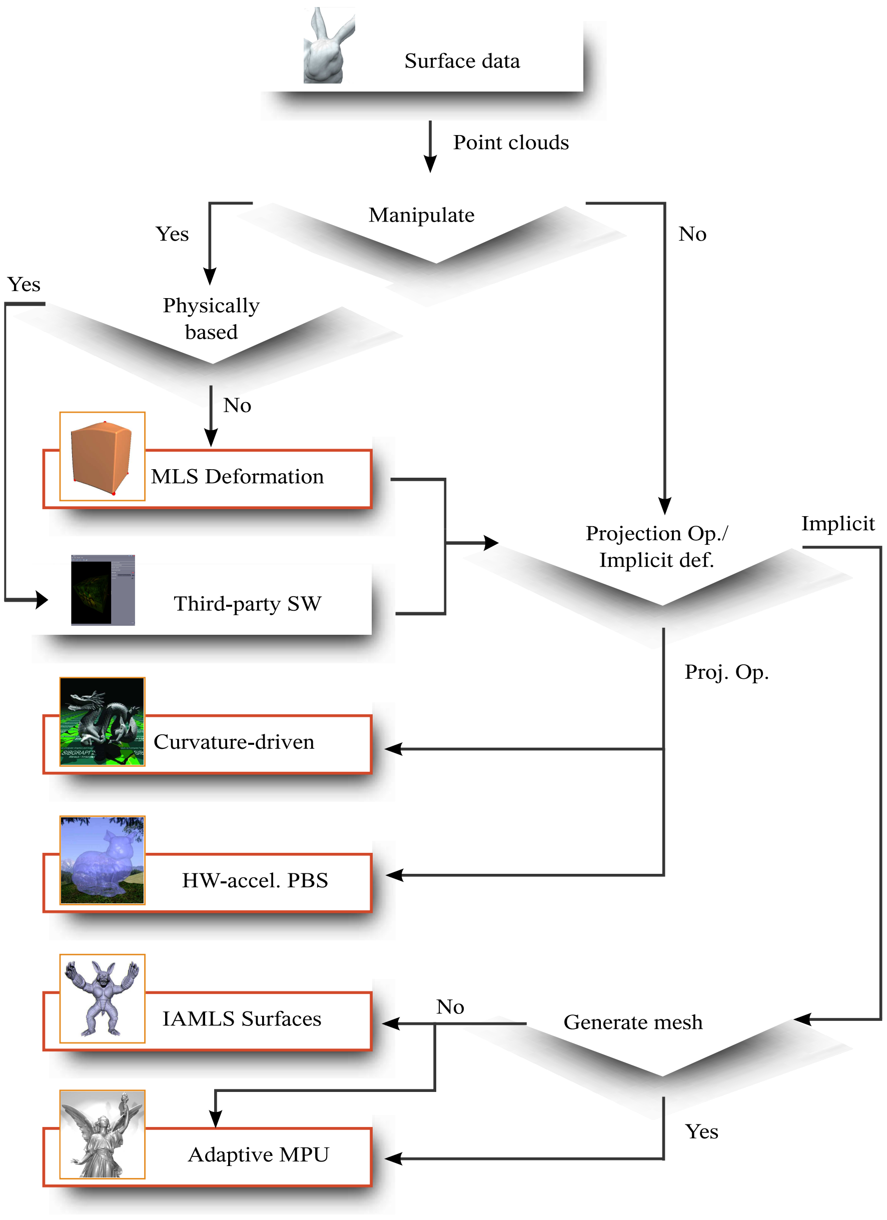


Figure C.20: Guideline for the use of a (incomplete) set of meshless techniques in the context of surface approximation. The methods proposed in this thesis are marked with red borders.

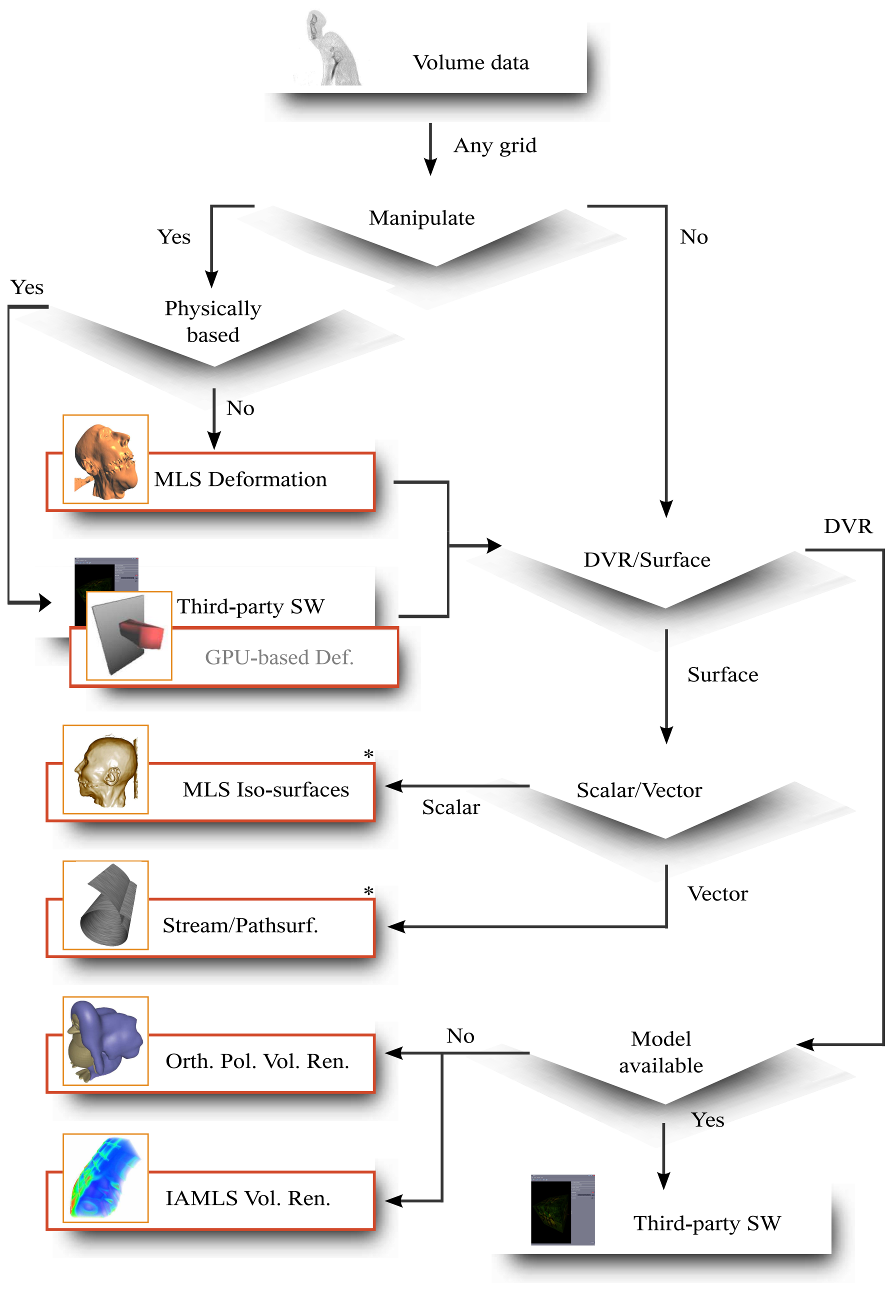


Figure C.21: Guideline for the use of a (incomplete) set of meshless techniques in the context of volume visualization. The methods proposed in this thesis are marked with red borders.

BIBLIOGRAPHY

[1] Bart Adams and Philip Dutré. Interactive boolean operations on surfel-bounded solids. *ACM Transactions on Graphics*, 22(3):651–656, 2003.

[2] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In *Proceedings of Eurographics/ACM Symposium on Geometry Processing*, pages 230–239. Eurographics Association, 2003.

[3] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In *Proceedings of the Shape Modeling International*, page 272, Washington, DC, USA, 2003. IEEE Computer Society.

[4] Gady Agam and Xiaojing Tang. A sampling framework for accurate curvature estimation in discrete surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):573–583, 2005.

[5] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.

[6] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proceedings of the ACM SIGGRAPH*, pages 157–164, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[7] Peter Alfeld. Scattered data interpolation in three or more variables. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 1–34. Academic Press, 1989.

[8] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the SIGGRAPH*, pages 415–421, New York, NY, USA, 1998. ACM Press.

[9] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proceedings of the ACM symposium on Solid modeling and applications*, pages 249–266, New York, NY, USA, 2001. ACM Press.

[10] Nina Amenta and Yong J. Kil. The domain of a point set surfaces. In *Proceedings of the Eurographics Symposium on Point-based Graphics*, pages 139–147. Eurographics Association, 2004.

[11] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. *ACM Transactions on Graphics*, 23(3):264–270, 2004.

[12] Erik Anderson, Steven Callahan, Carlos Scheidegger, John Schreiner, and Claudio Silva. Hardware-Assisted Point-Based Volume Rendering of Tetrahedral Meshes. In *Proceedings of SIBGRAPI*, pages 163–170. IEEE CS, 2007.

[13] K. S. Arun, Thomas S. Huang, and Steven D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):698–700, 1987.

[14] Anthony Atkinson and Marco Riani. *Robust diagnostic Regression Analysis*. Springer-Verlag, New York, 2000.

[15] Chandrajit L. Bajaj, Fausto Bernardini, and Guoliang Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proceedings of the SIGGRAPH*, pages 109–118, New York, NY, USA, 1995. ACM Press.

[16] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of ACM SIGGRAPH*, pages 43–54, 1998.

[17] Alan H. Barr. Global and local deformations of solid primitives. In *Proceedings of the ACM SIGGRAPH*, pages 21–30, New York, NY, USA, 1984. ACM.

[18] Richard H. Bartels and John J. Jezioranski. Least-squares fitting using orthogonal multinomials. *ACM Transactions on Mathematical Software*, 11(3):201–217, 1985.

[19] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[20] Ake Björck. *Numerical Methods for Least Squares Problems*. SIAM, London, U.K., 1996.

[21] Jules Bloomenthal. An implicit surface polygonizer. In Paul Heckbert, editor, *Graphics Gems IV*, pages 324–349. Academic Press, Boston, 1994.

[22] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[23] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schröder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Transactions on Graphics*, 22(3):917–924, 2003.

[24] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. PWS Publishing Co., Boston, MA, USA, 4th edition, 1989.

[25] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the Symposium on Volume Visualization*, pages 91–98, New York, NY, USA, 1994. ACM Press.

[26] Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. In *Proceedings of ACM SIGGRAPH*, pages 263–270, 1993.

[27] Wenli Cai and Pheng-Ann Heng. Principal stream surfaces. In *Proceedings of IEEE Visualization*, pages 75–80, 1997.

[28] Manfredo P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

[29] Antonio Castelo, Luís Gustavo Nonato, M.F. Siqueira, Rosane Minghim, and G. Tavares. The j_a^1 triangulation: An adaptive triangulation in any dimension. *Computer & Graphics*, 30(5):737–753, 2006.

[30] CD-ADAPCO. http://www.cd-adapco.com, 2007.

[31] Yan Chen, Qing hong Zhu, A. Kaufman, and Shigeru Muraki. Physically-based animation of volumetric objects. In *Proceedings of the Computer Animation*, page 154, Washington, DC, USA, 1998. IEEE Computer Society.

[32] C. S. Co, S. D. Porumbescu, and K. I. Joy. Meshless isosurface generation from multiblock data. In *Proceedings of Eurographics/IEEE TCVG Symposium on Visualization*, pages 273–281. Eurographics Association, 2004.

[33] C. S. Co, S. D. Porumbescu, and K. I. Joy. Meshless isosurface generation from multiblock data. In *Proceedings of Eurographics/IEEE TCVG Symposium on Visualization VisSym*, pages 273–281, 2004.

[34] Christopher S. Co, Bernd Hamann, and Ken I. Joy. Iso-splatting: A point-based alternative to isosurface visualization. In J. Rokne, W. Wang, and R. Klein, editors, *Proceedings of Pacific Graphics*, pages 325–334, 2003.

[35] Christopher S. Co and Ken I. Joy. Isosurface generation for large-scale scattered data visualization. In *Proceedings of Vision, Modeling, and Visualization*, pages 233–240, 2005.

[36] Carlos D. Correa, Deborah Silver, and Min Chen. Discontinuous Displacement Mapping for Volume Graphics . In *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization*, pages 9–16, 2006.

[37] Alvaro Cuno, Claudio Esperança, Antonio Oliveira, and Paulo Roma. 3D as-rigid-as-possible deformations using MLS. In *Proceedings of Computer Graphics International*, pages –, 2007.

[38] Philips J. Davis. *Interpolation and Approximation*. Blaisdell, New York, 1 edition, 1963.

[39] Carl de Boor. B-from basics. Technical summary report, Wisconsin Univ-Madison Mathematics Research Center, 1986.

[40] Tamal K. Dey and Jian Sun. An adaptive MLS surface for reconstruction with guarantees. In *Proceedings of the Eurographics Symposium on Geometry Processing*, page 43, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[41] Huong Quynh Dinh, Greg Turk, and Greg Slabaugh. Reconstructing surfaces using anisotropic basis functions. In *Proceedings of the International Conference on Computer Vision*, pages 606–613, 2001.

[42] Jean Duchon. Splines minimizing rotation-invariant seminorms in sobolev spaces. *Constructive Theory of Functions of Several Variables, Lecture Notes in Mathematics*, 571:85–100, 1977.

[43] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[44] Michael Elad. On the origin of the bilateral filter and ways to improve it. *IEEE Transactions on Image Processing*, 11(10):1141–1151, 2002.

[45] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Christof Rezk-Salama, and Daniel Weiskopf. *Real-Time Volume Graphics*. Ak Peters, 2006.

[46] Gregory Fasshauer. Approximate moving least-squares approximation: A fast and accurate multivariate approximation method. In *Curve and Surface Fitting*, pages 139–148, Saint-Malo, 2002. Nashboro Press.

[47] Gregory Fasshauer. Toward approximate moving least squares approximation with irregularly spaced centers. *Computer Methods in Applied Mechanics & Engineering*, 193:1231–1243, 2004.

[48] Gregory Fasshauer and Jack Zhang. Iterated approximate moving least squares approximation. In C. Alves V. M. A. Leitao and C. A. Duarte, editors, *Advances in Meshfree Techniques*, page to appear. Springer-Verlag, 2007.

[49] Markus Fenn and Gabrielle Steidl. *Robust local approximation of scattered data*, volume 31 of *Computational Imaging and Vision*, pages 317–334. Springer-Verlag, Dordrecht, 2005.

[50] Markus Fenn and Gabrielle Steidl. *Robust local approximation of scattered data*, volume 31 of *Computational Imaging and Vision*, pages 317–334. Springer-Verlag, 2005.

[51] Shachar Fleishman, Daniel Cohen-Or, and Claudio T. Silva. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3):544–552, 2005.

[52] Michael S. Floater, Géza Kós, and Martin Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005.

[53] Thomas Frühauf. Raycasting vector fields. In *Proceedings of IEEE Visualization*, pages 115–120, 1996.

[54] Michael P. Garrity. Raytracing irregular volume data. In *Proceedings of the Workshop on Volume Visualization*, pages 35–40, New York, NY, USA, 1990. ACM Press.

[55] Christoph Garth, Xavier Tricoche, Tobias Salzbrunn, Tom Bobach, and Gerik Scheuermann. Surface techniques for vortex visualization. In *Proceedings of EG/IEEE VGTC Symposium on Visualization*, pages 155–164, 2004.

[56] Joachim Georgii and Rüdiger Westermann. Mass-spring systems on the GPU. *Simulation Modelling Practice and Theory*, 13(8):693–702, 2005.

[57] Sarah F. Gibson. 3D chainmail: a fast algorithm for deforming volumetric objects. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 149–ff., New York, NY, USA, 1997. ACM.

[58] Joao Paulo Gois, Valdecir Polizelli-Junior, Tiago Etiene, Eduardo Tejadaand Antonio Castelo, Thomas Ertl, and Luis G. Nonato. Robust and Adaptive Surface Reconstruction using Partition of Unity Implicit. In *Proceedings of SIBGRAPI*, pages 95–104. IEEE CS, 2007.

[59] Joao Paulo Gois, Eduardo Tejada, Tiago Etiene, Luis G. Nonato, Antonio Castelo, and Thomas Ertl. Curvature-driven Modeling and Rendering of Point-Based Surfaces. In *Proceedings of the Brazilian Symposium on Computer Graphics and Image Processing*, pages 27–36. IEEE CS, 2006.

[60] Gene Golub and Charles Van Loan. *Matrix Computations*. John Hopkins Press, 1989.

[61] Markus H. Gross, Lars Lippert, R. Dittrich, and S. Häring. Two methods for wavelet-based volume rendering. *Computers and Graphics*, 21(2):237–252, 1997.

[62] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. In *Proceedings of ACM SIGGRAPH*, page 23, New York, NY, USA, 2007. ACM.

[63] Xiaohu Guo, Jing Hua, and Hong Qin. Touch-based haptics for interactive editing on point set surfaces. *IEEE Computer Graphics Applications*, 24(6):31–39, 2004.

[64] Roland L. Hardy. Multiquadric equations of topography and other irregular surfaces. *Journal of Geophysical Research*, 76:1905–1915, 1971.

[65] Mark Harris and Greg James. Physically-based simulation on graphics hardware, 2003. http://developer.nvidia.com/docs/IO/8230/GDC2003_PhysSimOnGPUs.pdf.

[66] Mark J. Harris, Greg Coombe, Thorsten Scheuermann, and Anselmo Lastra. Physically-based visual simulation on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 109–118, 2002.

[67] John C. Hart. Ray tracing implicit surfaces. Technical report EECS-93-014, Washington State University, 1993.

[68] Charles Hirsch. *Numerical Computational of Internal and External Flows*, volume 1. A Wiley-Interscience Publication, 1989.

[69] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the SIGGRAPH*, pages 71–78, New York, NY, USA, 1992. ACM Press.

[70] Berthold Horn. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, 5(7):1127–1135, 1987.

[71] Berthold Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 5(4):629–642, 1987.

[72] Jianbing Huang and Chia-Hsiang Menq. Combinatorial manifold reconstruction and optimization from unorganized point cloud with arbitrary topology. *Computer-Aided Design*, 1(34):149–165, 2002.

[73] Jeff P. Hultquist. Constructing stream surfaces in steady 3D vector fields. In *Proceedings of IEEE Visualization*, pages 171–178, 1992.

[74] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. In *Proceedings of the ACM SIGGRAPH*, pages 1134–1142, New York, NY, USA, 2005. ACM.

[75] Yun Jang, Ralf P. Botchen, Andreas Lauser, David S. Ebert, Kelly P. Gaither, and Thomas Ertl. Enhancing the Interactive Visualization of Procedurally Encoded Multifield Data with Ellipsoidal Basis Functions. In *Proceedings of Eurographics*, page 587. Eurographics Association, 2006.

[76] Yun Jang, Manfred Weiler, Matthias Hopf, Jingshu Huang, David S. Ebert, Kelly P. Gaither, and Thomas Ertl. Interactively Visualizing Procedurally Encoded Scalar Fields. In *Proceedings of Eurographics/IEEE TCVG Symposium on Visualization VisSym*, pages 35–44. Eurographics Association, 2004.

[77] Bruno Jobard, Gordon Erlebacher, and M. Youssuff Hussaini. Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):211–222, 2002.

[78] Thouis R. Jones, Fredo Durand, and Matthias Zwicker. Normal improvement for point rendering. *IEEE Computer Graphics and Applications*, 24(4):53–56, 2004.

[79] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of Eurographics Symposium on Geometry Processing*, pages 61–70. Eurographics Association, 2006.

[80] Richard Keiser, Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Contact handling for deformable point-based objects. In *Proceedings of Vision, Modeling, Visualization*, pages 339–347, 2004.

[81] Joe Kniss, Gordon L. Kindlmann, and Charles D. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization*, pages 255–262, 2001.

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

[83] Ravikrishna Kolluri. Provably good moving least squares. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 1008–1017, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[84] Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 11–21, New York, NY, USA, 2004. ACM Press.

[85] Martin Kraus. *Direct Volume Visualization of Geometrically Unpleasant Meshes*. PhD thesis, University of Stuttgart, 2003.

[86] Jens Krüger and Rüdiger Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics*, 22(3):908–916, 2003.

[87] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the SIGGRAPH*, pages 451–458, New York, NY, USA, 1994. ACM Press.

[88] P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 37(155):141–158, 1981.

[89] Peter Lancaster. *Polynomial and Spline Approximation*, chapter Moving weighted least-squares methods, pages 103–120. Reidel, 1979.

[90] Carsten Lange and Konrad Polthier. Anisotropic smoothing of point sets. *Computer Aided Geometry Design*, 22(7):680–692, 2005.

[91] Robert Laramee and R. Daniel Bergeron. An isosurface continuity algorithm for super adaptive resolution data. In *Proceedings of Computer Graphics International*, pages 215–237, 2002.

[92] Robert S. Laramee, Christoph Garth, J. Schneider, and Helwig Hauser. Texture advection on stream surfaces: A novel hybrid visualization applied to CFD simulation results. In *Proceedings of EG/IEEE VGTC Symposium on Visualization*, pages 155–162, 2006.

[93] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):143–161, 2004.

[94] Robert S. Laramee, Bruno Jobard, and Helwig Hauser. Image space based visualization of unsteady flow on surfaces. In *Proceedings of IEEE Visualization*, pages 131–138, 2003.

[95] Jinho Lee, Lance C. Burton, Raghu Machiraju, and Donna S. Reese. Efficient rendering of multiblock curvilinear grids with complex boundaries: Research articles. *Comput. Animat. Virtual Worlds*, 16(1):53–68, 2005.

[96] Seungyong Lee, George Wolberg, and Sung Yong Shin. Scattered data interpolation with multilevel b-splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):228–244, 1997.

[97] David Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67(224):1517–1531, 1998.

[98] David Levin. Mesh-independent surface interpolation. In Guido Brunnett, Bernd Hamann, Heinrich Müller, and Lars Linsen, editors, *Geometric Modeling for Scientific Visualization*, pages 37–49. Springer-Verlag, 2003.

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

[100] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital Michelangelo project: 3D scanning of large statues. In *SIGGRAPH ’00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[101] Marc Levoy and Turner Whitted. The use of points as a display primitive. Technical report, University of North Carolina at Chapel Hill, 1985.

[102] Guo-Shi Li, Xavier Tricoche, and Charles Hansen. GPUFLIC: Interactive and accurate dense visualization of unsteady flows. In *Proceedings of EG/ IEEE VGTC Symposium on Visualization*, pages 29–34, 2006.

[103] Yaron Lipman, Daniel Cohen-Or, and David Levin. Error bounds and optimal neighborhoods for MLS approximation. In *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 71–80, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[104] Yaron Lipman, Daniel Cohen-Or, and David Levin. Data-dependent MLS for faithful surface approximation. In *Proceedings of Eurographics Symposium on Geometry Processing*, pages 59–67, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[105] Yaron Lipman, Daniel Cohen-Or, David Levin, and Hillel Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Transactions on Graphics*, 26(3):22, 2007.

[106] Zhang Liu and Robert Moorhead. AUFLIC: An accelerated algorithm for unsteady flow line integral convolution. In *Proceedings of EG/IEEE TCVG Symposium on Visualization*, pages 43–52, 2002.

[107] Yarden Livnat and Xavier Tricoche. Interactive point-based isosurface extraction. In *Proceedings of the IEEE Conference on Visualization*, pages 457–464, Washington, DC, USA, 2004. IEEE Computer Society.

[108] Jean-Louis Maltret and Marc Daniel. Discrete curvatures and applications: a survey. Technical Report LSIS.RR.2002.002, Laboratoire des Sciences de l'Information et des Systèmes, 2002.

[109] Stephen R. Marschner and Richard J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of the IEEE Conference on Visualization*, pages 100–107, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[110] Dimitri J. Mavriplis. Revisiting the least-squares procedure for gradient reconstruction on unstructured meshes. Technical Report CR-2003-212683, NASA, 2006.

[111] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[112] Vladimir Maz'ya. Approximate approximation, in the mathematics of finite elements and applications. *Highlights*, 77, 1994.

[113] Boris Mederos, Sueni Arouca, Marcos Lage, Helio Lopes, and Luiz Velho. Improved partition of unity implicit surface reconstruction. Technical Report TR-0406, IMPA, Brazil, 2006.

[114] Boris Mederos, Luis Velho, and Luis Henrique Fiqueiredo. Robust smoothing of noisy point clouds. In *SIAM Conference on Geometric Design and Computing*, Seatle, 2003. Nashboro Press.

[115] Miriah Meyer, Robert M. Kirby, and Ross Whitaker. Topology, accuracy, and quality of isosurface meshes using dynamic particles. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1704–1711, 2007.

[116] Miriah Meyer, Blake Nelson, Robert M. Kirby, and Ross Whitaker. Particle systems for efficient and accurate high-order finite element visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1015–1026, 2007.

[117] Torsten Möller, Raghu Machiraju, Klaus Mueller, and Roni Yagel. Evaluation and design of filters using a Taylor series expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):184–199, 1997.

[118] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 49–54, 2002.

[119] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Transactions on Graphics*, 24(3):471–478, 2005.

[120] Matthias Müller, Richard Keiser, Andrew Nealen, Mark Pauly, Markus Gross, and Marc Alexa. Point based animation of elastic, plastic and melting objects. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, 2004.

[121] Matthias Müller, Matthias Teschner, and Markus Gross. Physically-based simulation of objects represented by surface meshes. In *Proceedings of Computer Graphics International*, pages 26–33, Washington, DC, USA, 2004. IEEE Computer Society.

[122] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics*, 22(3):463–470, 2003.

[123] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Sparse surface reconstruction with adaptive partition of unity and radial basis functions. *Graphical Models*, 68(1):15–24, 2006.

[124] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. Shape modeling with point-sampled geometry. *ACM Transactions on Graphics*, 22(3):641–650, 2003.

[125] Mark Pauly, Dinesh Pai, and Leonidas J. Guibas. Quasi-rigid objects in contact. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation*, pages 109–119, 2004.

[126] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: surface elements as rendering primitives. In *Proceedings of the SIGGRAPH*, pages 335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[127] Wilfried. Philips. Orthogonal base functions on a discrete two-dimensional region. Technical Report DG 91-20, ELIS, RUG, Universiteit Gent, 1992.

[128] PovRay. Persinstence of vision. http://www.porvay.org, 2007.

[129] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum*, volume 21, pages 461–470, 2002.

[130] Patrick Reuter, Pierre Joyot, Jean Trunzler, Tamy Boubekeur, and Christophe Schlick. Surface reconstruction with enriched reproducing kernel particle approximation. In *Eurographics Symposium on Point-Based Graphics*, pages 79–87. Eurographics Association, 2005.

[131] Christof Rezk-Salama, Michael Scheuering, Grzegorz Soza, and Günther Greiner. Fast volumetric deformation on general purpose hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 17–24, New York, NY, USA, 2001. ACM.

[132] Christian Rössl, Frank Zeilfelder, Günther Nürnberger, and Hans-Peter Seidel. Spline approximation of general volumetric data. In *Proceedings of the ACM Symposium on Solid Modeling and Applications*, pages 71–82, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[133] Miguel Sainz, Renato Pajarola, and Roberto Lairo. Points reloaded: Point-based rendering revisited. In *Proceedings of the Eurographics Symposium on Point-based Graphics*, pages 121–128. Eurographics Association, 2004.

[134] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-D shapes. In *Proceedings of ACM SIGGRAPH*, pages 197–206, 1990.

[135] Naohisa Sakamoto, Jorji Nonaka, Koji Koyamada, and Satoshi Tanaka. Volume rendering using tiny particles. In *Proceedings of the IEEE International Symposium on Multimedia*, pages 734–737, Washington, DC, USA, 2006. IEEE Computer Society.

[136] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Transactions on Graphics*, 25(3):533–540, 2006.

[137] Gunther Schmidt. Approximate approximations and their applications. In J. Rossmann, P. Takác, and G. Wildenhain, editors, *Operator Theory: Advances and Applications*, volume 109 of *The Maz'ya Anniversary Collection, v.1*, pages 111–136. Birkhäuser, 1999.

[138] John Schreiner, Carlos E. Scheidegger, Shachar Fleishman, and Claudio T. Silva. Direct (re)meshing for efficient surface processing. In *Proceedings of Eurographics*, pages 527–536, 2006.

[139] John Schreiner, Carlos E. Scheidegger, and Claudio T. Silva. High quality extraction of isosurfaces from regular and irregular grids. In *Proceedings of IEEE Visualization*, pages 1205–1212, 2006.

[140] Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredrick Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings of the IEEE Conference on Visualization*, pages 335–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[141] Chen Shen, James F. O'Brien, and Jonathan R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH*, pages 896–904, New York, NY, USA, 2004. ACM.

[142] Han-Wei Shen and D. L. Kao. A new line integral convolution algorithm for visualizing time-varying flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):98–108, 1998.

[143] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM Press.

[144] Jonathan Richard Shewchuck. What is a good linear element? Interpolation, conditioning, and quality measures. In *Eleventh International Meshing Roundtable*, pages 115–126, 2002.

[145] Jonathan R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

[146] Renben Shu, Chen Zhou, and Mohan S. Kankanhalli. Adaptive marching cubes. *The Visual Computer*, 11(4):202–217, 1995.

[147] Jasper V. Stokman, C. F. Dunkl, and Y. Xu. Orthogonal polynomials of several variables. *Approximation Theory*, 112(2):318–319, 2001.

[148] Gabriel Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *Proceedings of the International Conference on Computer Vision*, pages 902–907. IEEE Computer Society, 1995.

[149] Eduardo Tejada and Thomas Ertl. Large steps in GPU-based deformable bodies simulation. *Simulation Practice and Theory*, 13(9):703–715, 2005.

[150] Eduardo Tejada, Jõao P. Gois, Luis G. Nonato, Antonio Castelo, and Thomas Ertl. Hardware-accelerated extraction and rendering of point set surfaces. In *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization*, pages 21–28, 2006.

[151] Eduardo Tejada, Tobias Schafhitzel, and Thomas Ertl. Hardware-accelerated point-based rendering of surfaces and volumes. In *Proceedings of WSCG 2007 Full Papers*, pages 41–48, 2007.

[152] Matthias Teschner, Bruno Heidelberger, Matthias Müller, and Markus Gross. A versatile and robust model for geometrically complex deformable solids. In *Proceedings of the Computer Graphics International*, pages 312–319, 2004.

[153] Thomas Theußl, Torsten Möller, Jiří Hladůvka, and Meister Eduard Gröller. Reconstruction issues in volume visualization. Technical Report TR-186-2-01-14, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2001.

[154] Wai-Shun Tong and Chi-Keung Tang. Robust estimation of adaptive tensors of curvature by tensor voting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):434–449, 2005.

[155] Joost van de Weijer and Rein van den Boomgaard. Least squares and robust estimation of local image structure. *International Journal of Computer Vision*, 64(2/3):143–155, 2005.

[156] Jarke J. van Wijk. Flow visualization with surface particles. *IEEE Computer Graphics and Applications*, 13(4):18–24, 1993.

[157] Jarke J. van Wijk. Implicit stream surfaces. In *Proceedings of IEEE Visualization*, pages 245–252, 1993.

[158] Jarke J. van Wijk. Image based flow visualization. *ACM Transactions on Graphics*, 21(3):745–754, 2002.

[159] Jarke J. van Wijk. Image based flow visualization for curved surfaces. In *Proceedings of IEEE Visualization*, pages 123–130, 2003.

[160] Thomas Viklands. *Algorithms for Weighted Orthogonal Procrustes Problem and other Least Squares Problems*. PhD thesis, Department of Computing Science, Umeå University, Umeå, Sweden, 2006.

[161] Joachim Vollrath, Tobias Schafhitzel, and Thomas Ertl. Employing complex GPU data structures for the interactive visualization of adaptive mesh refinement data. In *Proceedings of the International Workshop on Volume Graphics*, pages 55–58, 2006.

[162] Ingo Wald and Hans-Petter Seidel. Interactive ray tracing of point-based models. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 1–8. Eurographics Association, 2005.

[163] Manfred Weiler, Ralf P. Botchen, Simon Stegmaier, Jingshu Huang, Yun Jang, David Ebert, Kelly Gaither, and Thomas Ertl. Hardware-assisted feature analysis and visualization of procedurally encoded multifield volumetric data. *Computer Graphics and Applications*, pages 72–81, 2005.

[164] Morris Weisfeld. Orthogonal polynomials in several variables. *Numerical Mathematics*, 1:38–40, 1959.

[165] Daniel Weiskopf and Gordon Erlebacher. Overview of flow visualization. In Charles. D. Hansen and Christopher R. Johnson, editors, *The Visualization Handbook*, pages 261–278. Elsevier, Amsterdam, 2005.

[166] Daniel Weiskopf and Thomas Ertl. A hybrid physical/device-space approach for spatio-temporally coherent interactive texture advection on curved surfaces. In *Proceedings of Graphics Interface*, pages 263–270, 2004.

[167] Eric W. Weisstein. Laguerre polynomial. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/LaguerrePolynomial.html, 2007.

[168] Holger Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, 1995.

[169] Rüdiger Westermann, Leif Kobbelt, and Thomas Ertl. Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[170] Rüdiger Westermann and Christof Rezk-Salama. Real-time volume deformations. *Computer Graphics Forum*, 20(3):–, 2001.

[171] Lee Westover. Footprint evaluation for volume rendering. In *Proceedings of the ACM SIGGRAPH*, pages 367–376, New York, NY, USA, 1990. ACM Press.

[172] Martin Wicke, Matthias Teschner, and Markus Gross. CSG-tree rendering for point-sampled objects. In *Proceedings of the Pacific Graphics*, pages 160–168. IEEE CS, 2004.

[173] Gernot Ziegler, Art Tevs, Christian Theobalt, and Hans-Peter Seidel. On-the-fly point clouds through histogram pyramids. In *Proceedings of Workshop on Vision, Modeling, and Visualization*, pages 137–144, 2006.