Technical Report

UCAM-CL-TR-658
ISSN 1476-2986

Number 658

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Intrinsic point-based surface processing

Carsten Moenning

January 2006

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted January 2005 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Queens' College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Summary

The need for the processing of surface geometry represents an ubiquitous problem in computer graphics and related disciplines. It arises in numerous important applications such as computer-aided design, reverse engineering, rapid prototyping, medical imaging, cultural heritage acquisition and preservation, video gaming and the movie industry. Existing surface processing techniques predominantly follow an extrinsic approach using combinatorial mesh data structures in the embedding Euclidean space to represent, manipulate and visualise the surfaces. This thesis advocates, firstly, the intrinsic processing of surfaces, i.e. processing directly across the surface rather than in its embedding space. Secondly, it continues the trend towards the use of point primitives for the processing and representation of surfaces.

The discussion starts with the design of an intrinsic point sampling algorithm template for surfaces. This is followed by the presentation of a module library of template instantiations for surfaces in triangular mesh or point cloud form. The latter is at the heart of the intrinsic meshless surface simplification algorithm also put forward. This is followed by the introduction of intrinsic meshless surface subdivision, the first intrinsic meshless surface subdivision scheme and a new method for the computation of geodesic centroids on manifolds. The meshless subdivision scheme uses an intrinsic neighbourhood concept for point-sampled geometry also presented in this thesis. Its main contributions can therefore be summarised as follows:

- An intrinsic neighbourhood concept for point-sampled geometry.
- An intrinsic surface sampling algorithm template with sampling density guarantee.
- A modular library of template instantiations for the sampling of planar domains and surfaces in triangular mesh or point cloud form.
- A new method for the computation of geodesic centroids on manifolds.
- An intrinsic meshless surface simplification algorithm.
- The introduction of the notion of intrinsic meshless surface subdivision.
- The first intrinsic meshless surface subdivision scheme.

The overall result is a set of algorithms for the processing of point-sampled geometry centering around a generic sampling template for surfaces in the most widely-used forms of representation. The intrinsic nature of these point-based algorithms helps to overcome limitations associated with the more traditional extrinsic, mesh-based processing of surfaces when dealing with highly complex point-sampled geometry as is typically encountered today.

Contents

List of Figures **9**

List of Tables **12**

List of Symbols and Abbreviations **14**

1 Introduction **17**

1.1 Background 19

1.2 Research motivation and objectives 21

1.3 Contributions 24

1.4 Thesis structure 25

2 Preliminaries **27**

2.1 Extrinsic vs. intrinsic distance mapping and geodesics 27

2.2 Geodesic Voronoi diagrams 29

2.3 Fast Marching level set methods 30

2.3.1 Fast Marching on Cartesian grids 30

2.3.2 Fast Marching for triangulated surfaces 35

2.3.3 Fast Marching for surfaces in point cloud and implicit form 39

2.4 Neighbourhood concepts for point-sampled geometry 42

2.4.1 Pitfalls for point-based neighbourhood concepts 43

2.4.2 Conventional extrinsic neighbourhood concepts 44
2.4.3 An intrinsic neighbourhood concept 47
2.5 Implementation details 48
2.5.1 Fast Marching on Cartesian grids 48
2.5.2 Fast Marching for surfaces in point cloud form 52
2.5.3 Geodesic Voronoi diagrams 53
2.6 Summary and discussion 54

3 Intrinsic point sampling of surfaces 57

3.1 Related Work 58
3.2 Farthest point sampling 60
3.3 The generic intrinsic point sampling algorithm 62
3.4 Modular algorithm structure 65
3.5 Sampling density guarantee 65
3.6 FastFPS of planar domains 67
3.6.1 Implementation of the generic algorithm for planar domains 67
3.6.2 Computational complexity 69
3.6.3 Analysis of the generated sampling distribution 70
3.6.4 Experimental results 73
3.7 FastFPS of triangulated surfaces 78
3.7.1 Implementation of the generic algorithm for triangulated surfaces . 79
3.7.2 Computational complexity 82
3.7.3 Experimental results 82
3.8 Summary and discussion 85

4 Intrinsic meshless surface simplification 89

4.1 Related Work 90

4.2 Intrinsic point cloud subsampling 92
4.3 Intrinsic point cloud resampling 94
4.3.1 Moving Least Squares 95
4.3.2 Determination of offset ball radii and hole-filling 99
4.4 Experimental results 100
4.5 Implementation details 103
4.5.1 Moving Least Squares 104
4.5.2 Enhanced k nearest neighbourhood 106
4.6 Summary and discussion 108

5 Intrinsic meshless surface subdivision 115

5.1 Related work 116
5.2 Intrinsic meshless surface subdivision 119
5.2.1 An intrinsic meshless surface subdivision scheme 119
5.2.2 Computation of geodesic centroids on manifolds 122
5.3 Experimental results 123
5.4 Implementation details 128
5.4.1 Intrinsic natural neighbourhoods 128
5.4.2 Geodesic centroid computation 129
5.4.3 Orthogonal projection 130
5.5 Summary and discussion 133

6 Conclusion and future work 139

6.1 Principal contributions 139
6.2 Other results 141
6.3 Future work 141

A Glossary of terms **143**

B Concepts from computational geometry **147**

C Data sources **151**

Bibliography **151**

List of Figures

1.1 Point cloud examples 18

1.2 Surface processing pipeline 19

1.3 Topological artefacts 22

1.4 St. Matthew - An example for a massively complex geometric model 23

1.5 Geodesic vs. Euclidean distance on non-linear manifolds 24

2.1 Development of swallowtails in front propagation 31

2.2 Fast Marching categorisation of grid vertices 32

2.3 A Cartesian Fast Marching example: Equal distance contours in 2D 34

2.4 Metrication error of non-geometric, graph-based distance mapping algorithms 35

2.5 Fast Marching for triangulated surfaces 36

2.6 Characteristic curve of the Eikonal equation across an acute triangle 37

2.7 Derivation of a quadratic upwind scheme for Fast Marching for triangulated surfaces 38

2.8 Offset band visualisation 40

2.9 Pitfalls for point-based neighbourhood concepts 44

2.10 Extrinsic neighbourhood concepts for point clouds 45

2.11 Conventional vs. enhanced k nearest neighbourhoods 46

2.12 Intrinsic natural neighbourhoods 47

2.13 Overlapping neighbour relations 48

2.14 Class design of my implementation of Fast Marching on Cartesian grids . . 49
2.15 Stencils supporting Fast Marching on Cartesian grids 50
2.16 Computation of discrete geodesic Voronoi diagrams 53

3.1 Computation of intrinsic farthest point samples 64
3.2 Refinement condition 64
3.3 Modular algorithm design 66
3.4 Voronoi vertex detection during front propagation 68
3.5 Partial intrinsic distance mapping 69
3.6 Point set distributions 70
3.7 Power spectral analysis 72
3.8 Adaptive FastFPS principle 72
3.9 Adaptive FastFPS for planar domains - Sample distributions 74
3.10 Adaptive FastFPS for planar domains - Image reconstructions 75
3.11 Uniform FastFPS for planar domains - Execution times 78
3.12 Terms used in update scheme of Fast Marching for triangulated surfaces . . 80
3.13 Uniform FastFPS of triangulated surfaces - Sample distribution 83
3.14 Uniform FastFPS of triangulated surfaces - Application example 83
3.15 Adaptive FastFPS of triangulated surfaces - Application example 83
3.16 Uniform FastFPS of triangulated surfaces - Level-of-details 84

4.1 Eigenanalysis for local normal estimation in Moving Least Squares 96
4.2 Moving Least Squares projection 97
4.3 Adaptive Moving Least Squares 98
4.4 Adaptive offset radii and hole-filling 100
4.5 Laser range scanning of a Buddha sculpture 102
4.6 Execution times of uniform intrinsic meshless surface simplification 104

4.7 Shape of covariance ellipsoids . . . 105
4.8 Computation of enhanced k nearest neighbourhoods . . . 107
4.9 Uniformly subsampled Michelangelo Day and Michelangelo Dawn point sets 111
4.10 Level-of-details of the Michelangelo Youthful point set . . . 112
4.11 Adaptive subsampling of the Venus point set using local surface variation estimates . . . 112
4.12 Adaptive subsampling of the Venus point set using a Moving Least Squares-based redundancy measure . . . 113
4.13 Analysis of the a posteriori approximation error . . . 113
4.14 Hole-filling of a Buddha point set . . . 114

5.1 Loop subdivision scheme for triangular control meshes . . . 117
5.2 Application example of Loop subdivision . . . 118
5.3 Geodesic vs. Euclidean and extrinsic centroids . . . 121
5.4 Spherical distance histograms . . . 125
5.5 Intrinsic natural neighbour detection during front propagation . . . 129
5.6 Approximately orthogonal projection . . . 132
5.7 Base point set and base mesh for subdivision of the unit sphere . . . 134
5.8 First iterations of intrinsic meshless and Loop subdivision of a sphere . . . 134
5.9 Second iterations of intrinsic meshless and Loop subdivision of a sphere . . 134
5.10 Meshless subdivision of the Michelangelo Youthful point set . . . 135
5.11 Meshless subdivision of the Michelangelo Youthful point set vs. the original data set . . . 136
5.12 Meshless subdivision of CAD model point sets . . . 136
5.13 Meshless subdivision of the Isis point set . . . 137

B.1 Medial axis of a surface . . . 148
B.2 Poles of a surface . . . 149

List of Tables

3.1 Peak signal-to-noise ratios 76

3.2 Uniform FastFPS for planar domains - Execution times 77

4.1 Execution times of uniform intrinsic meshless surface subsampling 103

4.2 Comparative evaluation of point cloud simplification algorithms 108

5.1 Uniform density of meshlessly subdivided point sets 126

5.2 Parameter settings for application examples 127

List of Symbols and Abbreviations

Roman Symbols

$B(p_i, r)$	Constant radius Euclidean offset ball around point $p_i \in P$.
$BS(p_i, p_j)$	Bisector of points p_i, $p_j \in P$, $p_i \neq p_j$.
$c_{\mathcal{N}_p}$	Weighted geodesic centroid of $\mathcal{N}_p$.
$d(\cdot, \cdot)$	Euclidean distance function.
$d_M(\cdot, \cdot)$	Intrinsic, or geodesic, distance function.
$D(p_i, p_j)$	Dominance region of p_i with respect to p_j, i.e. the region of M holding point p_i and bounded by $BS(p_i, p_j)$, p_i, $p_j \in P$, $p_i \neq p_j$.
eNN_p	Enhanced k nearest neighbourhood of point $p \in P$.
$F(\cdot)$	Propagation speed, or weight, function on M.
$\tilde{F}(\cdot)$	Propagation speed, or weight, function F smoothly extended into Ω_P^r or $\Omega_P^{r_i}$.
H	Local tangent plane approximation supporting MLS regressions.
l	Level of subdivision, $l \in \mathbb{Z}_0$.
m	Dimension of embedding (Euclidean) space of M. It is $m \geq 3$.
M	Differentiable, compact and connected Riemannian manifold in $\mathbb{R}^m$.
$\mathcal{N}_p$	Intrinsic natural, or Voronoi, neighbourhood of point $p \in P$.
$\mathcal{N}_{p_i p_j}$	Union of intrinsic natural neighbourhoods of points $p_i, p_j \in P$.
NN_p	Conventional k nearest neighbourhood of point $p \in P$.
P	Input point cloud acquired from M.
r	Constant radius of Euclidean balls centred at points $p_i \in P$.
r_i	Radius of Euclidean ball centred at point $p_i \in P$.

$R(p_i, P)$	Voronoi region of point $p_i \in P$ on M given P.
S	Set of point samples.
$T(\cdot)$	Weighted extrinsic arrival time function, also called weighted extrinsic distance map (function).
$T_M(\cdot)$	Weighted intrinsic arrival time function, also called intrinsic, or geodesic, distance map (function) on M.
$T_{\Omega_P^r}(\cdot)$	Weighted extrinsic arrival time function computed in Ω_P^r.
T_xM	Tangent space to M at $x \in M$.
$VD(P)$	Intrinsic, or geodesic, Voronoi diagram of P.

Greek Symbols

δ	Global parameter scaling the Gaussian weight function used in the context of MLS surface approximation.
Δ	Finite difference operator.
θ_j	Weight function determining the influence of point p_j in MLS surface approximation and in the orthogonal projection of a point p_i; $p_i, p_j \in P$, $p_i \neq p_j$; θ_j is assumed to be smooth, positive and monotonically decreasing with distance.
$\Pi_M(\cdot)$	Orthogonal projection operator from $\mathbb{R}^m$ onto M.
ρ	Refinement condition ($\rho > 0$).
τ_c	Radius of sphere enclosing the neighbourhood of point p_i, centred at the weighted Euclidean centroid of the neighbourhood.
τ_i	Radius of sphere enclosing the neighbourhood of point p_i, centred at p_i.
$\Upsilon_P(\cdot)$	MLS projection operator given P.
Ω_P^r	Constant radius Euclidean offset band centred a points $p_i \in P$.
$\Omega_P^{r_i}$	Variable radius Euclidean offset band centred a points $p_i \in P$.

Other Symbols

$\langle \cdot, \cdot \rangle$	Inner product.
$\| \cdot \|$	Norm of the object under consideration.
$\partial R(\cdot, \cdot)$	Boundary of Voronoi region $R(\cdot, \cdot)$.
∇	Euclidean vector differential operator.
∇_M	Intrinsic vector differential operator on M.

Abbreviations

FastFPS	Fast Marching farthest point sampling.
MLS	Moving Least Squares.
MSE	Mean-square (reconstruction) error.
PDE	Partial Differential Equation.
PSNR	Peak signal-to-noise ratio.
UML	Unified Modeling Language.

Chapter 1

Introduction

This thesis is concerned with the intrinsic processing of surfaces in Computer Graphics and related disciplines with particular focus on the intrinsic processing of meshless, point-based surface representations. Point-based surface representations consist of point samples which provide information on the geometry of the surface in three or higher dimensions and, possibly, its appearance. This set of point samples, also called point cloud, does not carry any connectivity (neighbouring) information and can be non-uniformly distributed in space. See Figure 1.1 for some examples. "Intrinsicness" is understood as the geometry of the surface from the point of view of the surface "inhabitants" who have no knowledge of the embedding space.

Since we are living in a 3D world, any object we consider features a surface, its boundary in the embedding space. It seems intuitively appealing to analyse and process surface data acquired from terrain, cars, artefacts, internal organs, etc. from the point of view of a being living in the embedding space, i.e. extrinsically. This approach has been adopted extensively and successfully for the solution of important problems such as continuous surface reconstruction and (mesh-based) surface simplification. See, e.g. Kobbelt et al. [86].

With the advent of increasingly detailed point-based surface representations, it is more natural, however, to operate with the raw data directly and intrinsically rather than attempting to extract a single continuous surface representation in the embedding space, a (surface reconstruction) process which tends to be complex and error-prone. Also, as recent advances in point-based surface processing have shown, topological information in the form of, for example, extrinsic mesh connectivity, is no longer needed to support a complete surface processing pipeline from data acquisition to surface rendering and point-based surface representations can be used throughout instead. See, e.g. Kobbelt

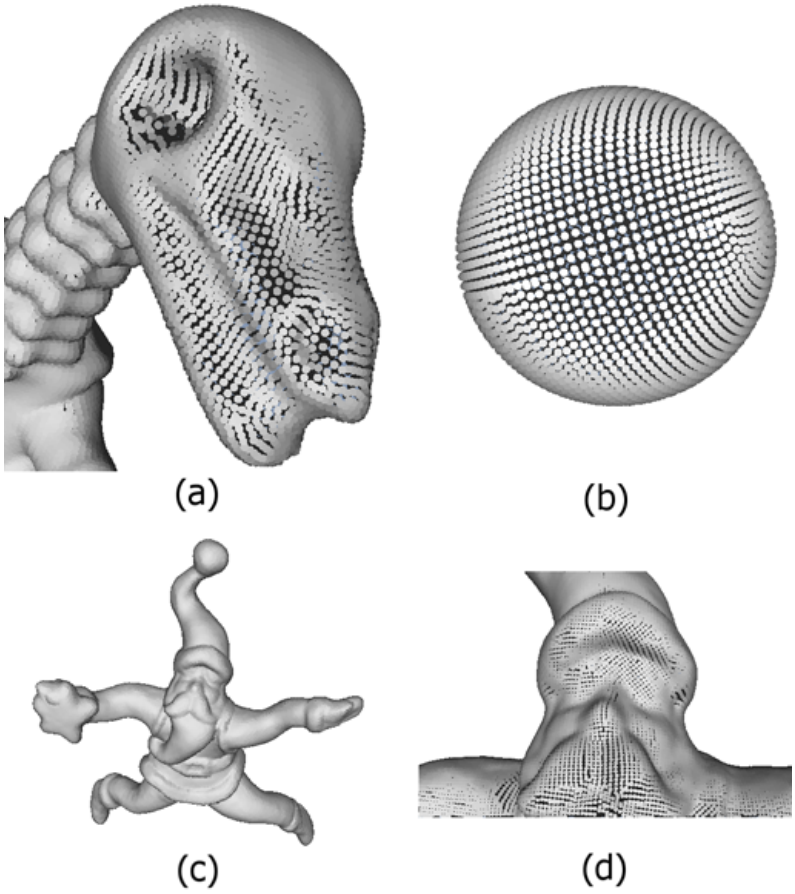

Figure 1.1: Examples for surfaces represented by point clouds. The dinosaur (a), sphere (b) and Santa (c) models are represented and rendered using point primitives in Pointshop3D [178]; (d) shows a zoomed-in view of the Santa model.

and Botsch [87].

This thesis advocates the intrinsic processing of surfaces in general and that of surfaces in point cloud form in particular. It is shown that important surface processing operations such as surface sampling, surface simplification and surface subdivision [44, 177] can be performed meshlessly, working efficiently with the point-sampled geometry directly and intrinsically.

The motivation for and the objectives of this research are best seen in the context of a surface processing pipeline, as typically encountered today. Section 1.1 describes the main aspects of such a processing pipeline and comments on its data acquisition, surface reconstruction, surface manipulation and rendering steps. This is followed by a discussion of the recent progress towards an exclusively point-based surface processing pipeline. Against this background, I set out the motivation for and the objectives of this thesis in Section 1.2. Its contributions are summarised in Section 1.3. The thesis structure presented in Section 1.4 concludes this chapter.

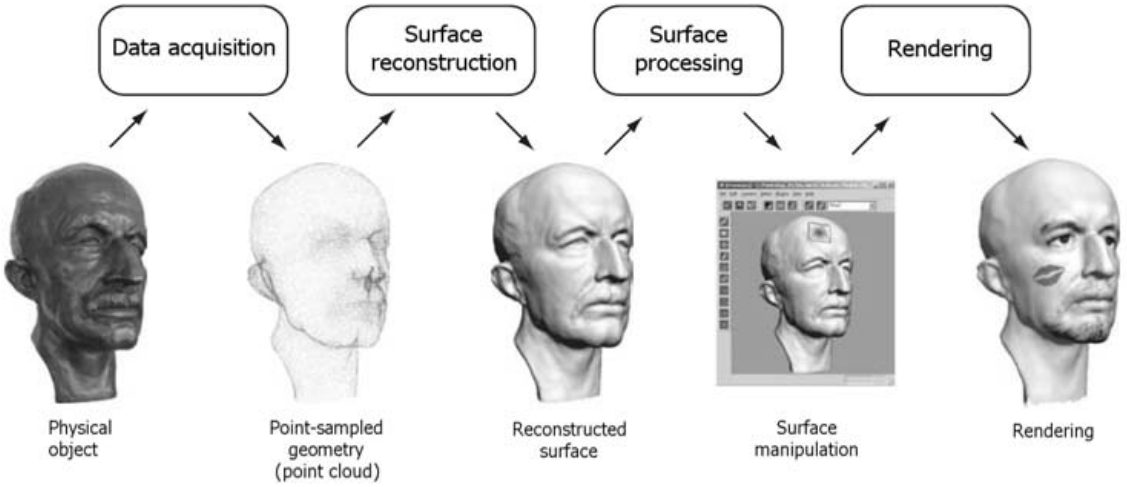

Figure 1.2: Example of a surface processing pipeline from data acquisition to rendering as typically encountered today. This example deals with the processing of surface data acquired from a physical object and is adapted from [122].

1.1 Background

To put the topic of this research into context, this section summarises surface processing as typically performed today. The various steps of this process can be seen in the form of a processing pipeline as illustrated in Figure 1.2 and are discussed in detail in the following.

Surface processing is initiated by the production of surface data, i.e. *data acquisition*. Surface data may be the result of an interactive model design process, experiments (natural sciences, engineering), computations (numerical simulations, evaluation of mathematical functions) or the measuring of relevant properties of a physical object. Due to the increasing availability of highly accurate and affordable acquisition devices, this latter source for surface data has gained substantial importance in fields such as CAD, reverse engineering, cultural heritage preservation, archaeology, eCommerce, animation, special effect generation, etc. The most widely-used acquisition devices use active optical sensing techniques such as laser range or structured light scanning and time-of-flight systems. Passive optical acquisition methods include shape-from-stereo, shape-from-shading, or shape-from-silhouettes. For details on the various acquisition techniques, see Scopigno et al. [141]. Irrespective of the acquisition technique used, the result of the data acquisition step is typically a discrete description of the object surface, i.e. a point cloud. Apart from its geometry, this set of point samples may also carry appearance attributes such as colour or texture.

Most existing algorithms for surface processing tasks such as surface subdivision [44,177],

free-form modelling [86] or surface rendering [168] assume (piecewise) continuous surface descriptions. During *surface reconstruction*, a continuous approximation of the unknown surface is therefore computed from the point samples returned by the data acquisition step. Numerous reconstruction techniques have been suggested in the recent past predominantly producing polygonal mesh [8–10, 15, 18, 19, 38, 45, 72] or implicit representations [25, 140, 175] of the underlying surface. For the mesh reconstruction case, recent advances in computational geometry allow the reconstruction to be performed with geometric and topological guarantees. That is, the reconstructed surface is guaranteed to be topologically equivalent to the original surface and to approximate it with a small geometric error [8–10, 45]. These results typically assume that the original surface represents a two-manifold in $\mathbb{R}^3$ which has been sufficiently densely sampled during the data acquisition step. The corresponding algorithms make use of higher-dimensional extrinsic combinatorial data structures such as the 3D Delaunay triangulation [120] of the point set, from which a subcomplex is extracted heuristically which is shown to intersect the underlying surface. The resulting mesh represents the final triangular surface representation used for further processing.

In the case of optically acquired surface data in particular, the frequently highly dense point cloud representations produced by the acquisition step are converted into highly complex meshes by these algorithms. As a consequence, *mesh simplification* may be required before further mesh processing tasks can be attempted. See, for example, Gotsman et al. [58] for a recent survey of mesh simplification techniques. Further surface processing may not only be hampered by the complexity of the mesh but also its poor quality following surface reconstruction. Poor quality expresses itself in the form of, for example, irregular triangle shapes, angles, sizes or connectivity. As a result, the mesh may be unsuitable for tasks such as finite element or numerical analysis (due to numerical instabilities) and surface subdivision. In the case of surface subdivision, mesh subdivision schemes assume semi-regular connectivity. For example, for a surface in $\mathbb{R}^3$, this requires a majority of mesh patches featuring vertices with exactly six outgoing edges and a small number of irregular vertices (with a number of outgoing edges different from six) connecting these patches [177]. *Remeshing* tries to address these problems by resampling the mesh subject to certain quality criteria. See Alliez et al. [7] for a review of remeshing techniques. Following these, frequently combined, simplification and remeshing procedures, the mesh representation is ready for meaningful further processing.

A surface processing pipeline is typically concluded with the *rendering* of the geometric model. Most modern graphics hardware is still optimised for the rendering of triangular meshes. Thus, even if the underlying surface was previously represented implicitly or

in the form of point samples, at the rendering stage, it will need to be converted into a triangular mesh using surface reconstruction. Depending on the quality required, the model rendering is then produced off-line using ray-tracing or, in the case of interactive applications, using the z-buffer. Watt and Watt [168] provide details on the various rendering techniques.

Thus, surface processing, as typically encountered today, follows a mesh-based, extrinsic processing approach. The following section discusses some of the problems associated with this approach and the benefits of performing surface processing intrinsically and point-based.

1.2 Research motivation and objectives

Surface processing today predominantly follows an extrinsic approach using combinatorial mesh data structures in the embedding Euclidean space to represent, manipulate and visualise the surface. This thesis advocates, firstly, processing directly across the surface, i.e. intrinsic processing. Secondly, it continues the trend towards the use of point primitives for the representation and processing of surfaces. It is hypothesised that as a result of the combination of these two notions, some of the limitations, discussed below, of the traditional mesh-based, extrinsic approach towards the processing of the kind of point set typically to be dealt with today can be avoided. The overall objective of this thesis may be stated as the development of algorithms for the intrinsic processing of surfaces in general and those represented in point cloud form in particular.

Such an objective is worthwhile to be pursued for a number of reasons. First of all, as indicated in the preceding section, extrinsic mesh-based surface processing, typically involves steps such as surface reconstruction, mesh simplification and remeshing. In the case of point-sampled geometry, the quality of the acquired point cloud is often affected by shortcomings of the acquisition process. When using optical sensing techniques, for example, data acquisition can be affected by measurement errors, the occlusion of parts of the object from the sensor, inadequate lighting conditions, etc. [141]. The surface reconstruction step is then likely to produce topological artefacts in the form of meshes of arbitrary connectivity and arbitrary genus, i.e. incorrectly introduced object handles (Figure 1.3). See also Wood et al. [173]. These artefacts cause numerical instabilities for applications such as finite element analysis. They also have the knock-on effect of inferior results from, for example, mesh simplification, remeshing and surface subdivision [173]. Also note that, irrespective of the quality of the acquired data, these mesh operations are

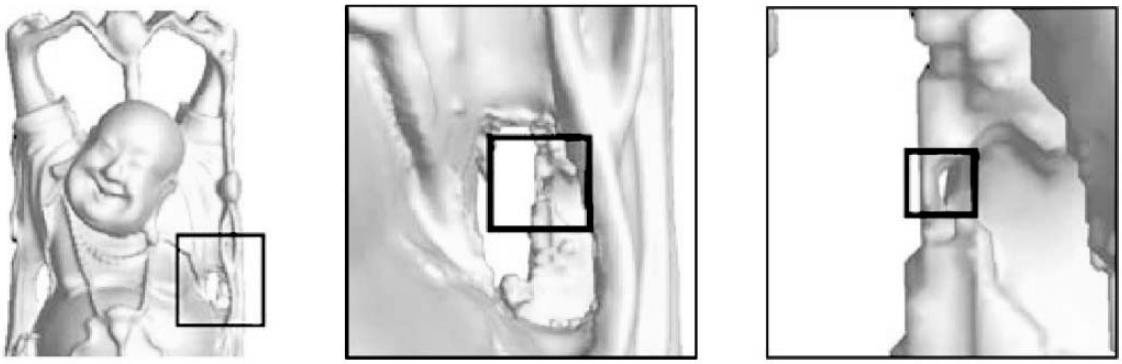

Figure 1.3: This sequence of close-ups reveals an object handle incorrectly introduced by surface reconstruction from point-sampled geometry featuring, for example, measurement errors and the occlusion of concave regions of the object. This example is adapted from Wood et al. [173].

generally costly both in terms of computational and memory demands. For example, those surface reconstruction algorithms which give geometric and topological guarantees (see, e.g. the work by Amenta et al. [8–10]), tend to be ill-equipped to deal with the millions of point samples routinely produced by modern data acquisition devices (see, e.g. Levoy et al. [94]) (Figure 1.4). The complexity of these methods when dealing with samples acquired from high-dimensional surfaces or surfaces of high co-dimension is prohibitive. This is due to the overhead and performance hits associated with mesh data structure maintenance and traversal in this setting. In the case of extremely high-dimensional manifolds by samples (see, e.g. Belkin and Niyogi [14] and Tenenbaum et al. [156]), mesh-based processing breaks down at the surface reconstruction step and any processing needs to deal with the raw point data instead. The advent of highly complex geometric models further implies that even if surface reconstruction was performed successfully, during mesh-based rendering, the screen space projection of the mesh elements will frequently occupy less space than a pixel. Thus, mesh-based rendering of complex geometric models becomes inefficient and the display of point primitives represents the more attractive alternative.

Point-based rendering allows to visualise complex geometric models output-sensitively and with relatively simple level-of-detail control [77, 101, 153]. Points, however, may not only be used as a display primitive [22, 59, 76, 78, 95, 129, 139, 179, 180] but also for editing and modeling purposes [5, 97, 125, 126, 178] and meshless surface approximation [1, 5]. Overall, the recent progress in the area of point-based geometry processing has been sufficiently substantial to support an exclusively point-based surface processing pipeline. The often complex, arbitrary and error-prone reconstruction of polygonal or implicit representations of the underlying surface is thereby avoided and all processing is performed

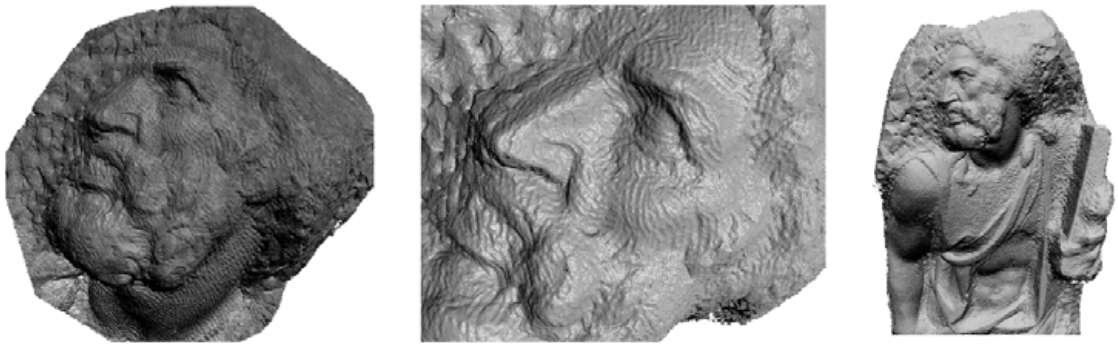

Figure 1.4: An example for a massively complex geometric model as typically produced today: Renderings of the Michelangelo St. Matthew surface reconstructed from a set of approximately 600 million points.

geometrically and with the raw data instead. Since point-based processing generally allows to represent objects without the need for the explicit description of their topology, it is particularly attractive for the processing of high-dimensional surfaces or surfaces of high co-dimension. The lack of connectivity information is similarly beneficial when it comes to the deformation of the surface representation. Since no topological information needs to be maintained, point-based surface deformation improves upon meshes in terms of editing flexibility [87].

It is attractive to combine these advantages of point-based surface processing with those inherent to the intrinsic processing of surfaces. Mesh processing algorithms, for example, generally compute distances between two points on a surface in the Euclidean metric. This only represents a good approximation to their true distance along the surface, i.e. their geodesic distance, in low curvature regions or for short distances. The Euclidean distance is linear in nature and thus simply cannot capture the geometric structure of non-linear manifolds. For the example shown in Figure 1.5, the Euclidean distance between the points is a straight line in the embedding space. The non-linear geometric structure of the manifold is ignored. By contrast, the geodesic distance is the shortest curve along the surface thereby capturing the non-linear structure of the object. The issues of, for example, arbitrary mesh connectivity and mesh distortions following surface reconstruction represent typical results of this inaccurate distance approximation when dealing with non-linear manifolds. The arbitrary nature of mesh connectivity in particular stems from the fact that it is not inherent to the geometry but artificially imposed by using a linear metric.

Intrinsic processing is performed on the surface and thus features the surface's dimensionality. When using extrinsic combinatorial data structures, the dimensionality of the problem coincides with the dimensionality of the embedding space. The higher the co-

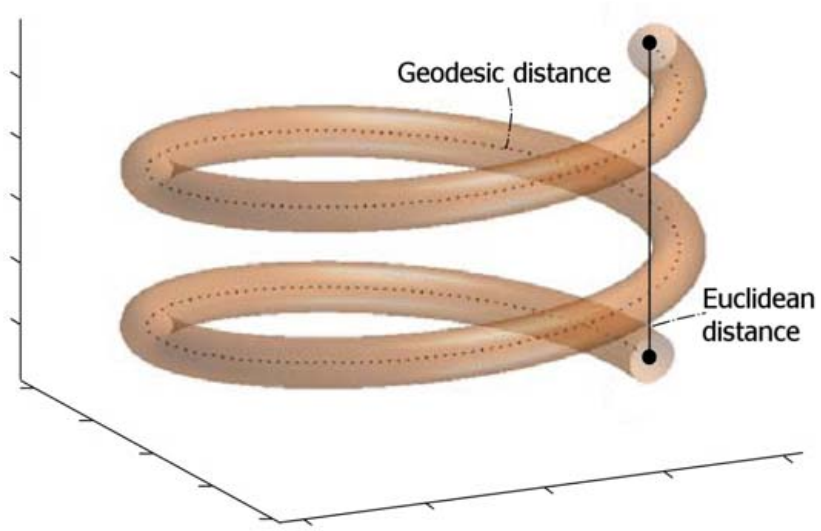

Figure 1.5: The Euclidean distance between two (black) points on this non-linear manifold ignores its geometric structure and is computed in the form of a straight line in the embedding space. By contrast, the geodesic distance is computed along the surface thereby taking its non-linearity into account. This example is adapted from [69].

dimension of the surface, the more preferable intrinsic processing therefore becomes. Nevertheless, techniques using Euclidean distances remain popular since it used to be relatively more difficult to compute geodesic distances instead. With the advent of Fast Marching level set methods for surfaces in triangular [84,146], implicit [103] or point cloud form [104], however, this is no longer the case. These methods permit the very efficient computation of geodesic distances for surfaces in the most widely-used forms of representation. It is the objective of this thesis to use these techniques for the design of algorithms operating truly intrinsically across surfaces. This theme is largely unexplored for surfaces in point cloud form in particular. My research aims at filling this gap by devising algorithms for intrinsic point-based surface processing. The particular contributions made in this context are listed in the following section.

1.3 Contributions

This thesis presents algorithms for the intrinsic processing of surfaces in general and that of point-based surface representations in particular. Its contributions can be summarised as follows:

- A new intrinsic neighbourhood concept for point-sampled geometry is introduced.
- An intrinsic surface sampling algorithm template with user-controlled, guaranteed

sampling density is proposed.

- A modular library of template instantiations for the point sampling of planar domains and surfaces in triangular mesh or point cloud form is presented.
- A new method for the computation of geodesic centroids on manifolds is put forward.
- The first intrinsic meshless surface simplification algorithm is presented.
- The notion of intrinsic meshless surface subdivision is proposed.
- The first intrinsic meshless surface subdivision scheme is introduced.

1.4 Thesis structure

The presentation is organised as follows:

- **Chapter 2** introduces notation and concepts used throughout this thesis. These include in particular the notion of intrinsic vs. extrinsic distance mapping, (minimal) geodesics, geodesic vs. extrinsic centroids, geodesic Voronoi diagrams and recent advances in Fast Marching level set methods. A new neighbourhood concept for point-sampled geometry is presented in the form of intrinsic natural neighbourhoods. Implementation details for those concepts used repeatedly in subsequent chapters are given.
- **Chapter 3** puts forward a generic intrinsic point sampling algorithm for planar domains and surfaces in triangular mesh, implicit and point cloud form. The theoretical concepts underpinning this algorithm are discussed in detail for the planar domain case. This is followed by an analysis and a number of application examples for the uniform and flexibly non-uniform sampling of images. The chapter is concluded with the instantiation of the algorithm template for surfaces in triangular mesh form and examples for its applications.
- **Chapter 4** presents my intrinsic surface simplification algorithm. The algorithm is experimentally and comparatively evaluated in the context of the uniform and non-uniform subsampling and resampling of surfaces represented in the form of massive point sets.
- **Chapter 5** introduces the notion of intrinsic meshless surface subdivision. The first intrinsic meshless surface subdivision scheme is presented and used for the

subdivision of a number of different surfaces in point cloud form. The results are analysed qualitatively and quantitatively.

- **Chapter 6** summarises the results of this research and concludes with directions for future work.

Chapter 2

Preliminaries

This thesis is concerned with the intrinsic processing of surfaces. In the following, a number of concepts from Riemannian geometry used throughout this thesis are therefore presented. These include fundamental ideas such as manifold surfaces, intrinsic (geodesic) vs. extrinsic distance and (minimal) geodesics (Section 2.1). Given an understanding of the notion of geodesic distance, it is then possible to define neighbourhood centroids (Section 2.1) and Voronoi diagrams (Section 2.2) in terms of intrinsic rather than extrinsic proximity. These concepts are used repeatedly in this thesis to take over the roles of their counterparts in extrinsic surface processing. The question of how to compute geodesic distances is addressed in Section 2.3 with a detailed discussion of Fast Marching level set methods for Cartesian grids [145], triangular meshes [84], implicit surfaces [103] and point clouds [104]. Since this research is predominantly concerned with surfaces in point cloud form, i.e. a surface representation which features no point neighbourhood information, Section 2.4 discusses neighbourhood concepts for point-sampled geometry and proposes a geodesic Voronoi diagram-based neighbourhood concept. The chapter concludes with details on my implementations of Fast Marching for Cartesian grids, triangular meshes and point clouds and geodesic Voronoi diagrams (Section 2.5).

2.1 Extrinsic vs. intrinsic distance mapping and geodesics

Let (surface) M be a differentiable (smooth), compact and connected Riemannian manifold in $\mathbb{R}^m$, $m \geq 3$. The term *intrinsic processing* refers to processing directly on M rather than in its embedding space. The Riemannian metric on M at point $x \in M$ is a smoothly varying inner product $\langle \cdot, \cdot \rangle$ on the tangent space T_xM. The norm of a vector v in T_xM is given by $\|v\| = \langle v, v \rangle^{\frac{1}{2}}$. M is endowed with the metric inherited from $\mathbb{R}^m$, hence

$\langle v, z \rangle$ will be the usual inner product for vectors v and z in $\mathbb{R}^m$. Consider a (sectionally) smooth curve $\gamma : [a, b] \subseteq \mathbb{R} \to M$ parameterised by $\mathtt{t}$. The length $\mathcal{L}(\gamma)$ of $\gamma(\mathtt{t})$ weighted by positive weight $w(\gamma(\mathtt{t}))$ follows from integrating the norm of its tangent vectors, $\dot{\gamma}(\mathtt{t})$, along the curve, i.e.

$$\mathcal{L}(\gamma) = \int_a^b w(\gamma(\mathtt{t}))\|\dot{\gamma}(\mathtt{t})\|\mathtt{dt}$$

The curve γ is called the (minimising) (w-)*weighted geodesic* from a point x to a point y on M, if it represents the minimum length-curve among all the curves on M joining x and y, where $\gamma(a) = x$ and $\gamma(b) = y$. Since we are assuming M to be compact, it is geodesically complete and there exists at least one such curve on M but it may not be unique. The length of the weighted geodesic between x and y gives the *weighted intrinsic*, or *weighted geodesic*, *distance*, $d_M(x, y)$, between the points, i.e.

$$d_M(x, y) = \inf_{\gamma}\{\mathcal{L}(\gamma)\}$$

The function giving the weighted intrinsic distance from a point $x \in M$ to every point in M, $d_M(x, \cdot)$, is called the *weighted intrinsic distance function*, or *weighted intrinsic distance map*, of x. Unique weighted geodesics between two points x and y on M may thus be computed from $d_M(x, \cdot)$ by backtracking from y towards x in the direction of the negative gradient of $d_M(x, \cdot)$, i.e. in the direction of steepest descent.

Partial weighted intrinsic distance mapping refers to the restriction of the extent of the distance map of a point x to points on M such that $d_M(x, \cdot) \leq d_M(y, \cdot)$, $x, y \in M$, $x \neq y$. That is, given a weighted intrinsic distance map $d_M(y, \cdot)$, the extent of $d_M(x, \cdot)$ will be limited to those points on M which are intrinsically at most as far away from x as they are from y.

The *weighted extrinsic distance* between points x and y on M, $d(x, y)$, is computed in the metric of the embedding space. Since this thesis is concerned with manifolds in $\mathbb{R}^m$, the weighted extrinsic distance is Euclidean and given by the length of the Euclidean line segment between x and y in the embedding space. Note that apart from its end points, this line segment generally does not lie on the manifold. In the simple case of M being a plane, however, the weighted intrinsic and weighted extrinsic (Euclidean) distance coincide and geodesics are straight lines. More detail on the above notions may be found in, for example, Chavel [28].

The term *weighted geodesic*, or *weighted intrinisic, centroid* refers to the mean of a local neighbourhood of points on M given by the minimiser of the weighted sum of squared

intrinsic distances between the points. This is to be distinguished from the *weighted extrinsic centroid* of the subset, computed using Euclidean distances in the embedding space and subsequently projected onto the manifold.[1]

2.2 Geodesic Voronoi diagrams

A Voronoi diagram partitions a domain into regions of closest neighbourhoods for a set of source points [120]. For the case of manifolds, consider a set of point samples $P = \{p_1, p_2, \ldots, p_n\}$ acquired without noise from M, i.e. $P \subset M$. Define the bisector $BS(p_i, p_j)$ of p_i, $p_j \in P$, $p_i \neq p_j$, as geodesically equidistant loci with respect to p_i, p_j, i.e.

$$BS(p_i, p_j) = \{q \in M : d_M(p_i, q) = d_M(p_j, q)\}$$

Let the dominance region of p_i with respect to p_j, $D(p_i, p_j)$, denote the region of M containing p_i bounded by $BS(p_i, p_j)$. The *Voronoi region* of p_i given P is given by

$$R(p_i, P) = \bigcap_{p_j \in P, p_j \neq p_i} D(p_i, p_j)$$

and consists of all points in M for which the geodesic distance to p_i is smaller than or equal to the geodesic distance to any other point $p_j \in P$. The boundary shared by a pair of Voronoi regions is called a *Voronoi edge*. Voronoi edges meet at *Voronoi vertices*. The *geodesic Voronoi diagram* of P is defined as

$$VD(P) = \bigcup_{p_i \in P} \partial R(p_i, P),$$

where $\partial R(p_i, P)$ denotes the boundary of $R(p_i, P)$.

The term *bounded* geodesic Voronoi diagram refers to the conjunction of $VD(P)$ with the domain. A bounded geodesic Voronoi diagram consists of bounded Voronoi regions. The set of vertices of a bounded $VD(P)$ includes points of intersection of Voronoi edges with the manifold boundary when dealing with an open manifold.

For further details on the notion of geodesic Voronoi diagrams, see Kunze et al. [88] and Leibon and Letscher [90].

[1] For notational convenience, if $w = 1$ across the domain or the distinction between weighted and unweighted distance mapping and weighted and unweighted centroids is immaterial in the given context, the "weighted"-qualifier will henceforth be dropped from the terms introduced.

Throughout this thesis, the continuous geodesic Voronoi diagram $VD(P)$ is approximated by its discrete counterpart using intrinsic distance mapping computed across a structured or unstructured grid. Intrinsic distance mapping may be performed very efficiently by using recent advances in Fast Marching level set methods reviewed next.

2.3 Fast Marching level set methods

Fast Marching level set methods represent very efficient techniques for the solution of front propagation problems which can be formulated as boundary value partial differential equations (PDEs). As discussed in the following, the problem of computing a point's intrinsic distance map across a surface may be posed in the form of such a partial differential equation. The conventional Fast Marching technique towards the approximation of its solution on Cartesian grids is presented in Section 2.3.1. Section 2.3.2 gives its extension to surfaces in triangulated form. Section 2.3.3 deals with the extension of the conventional Fast Marching technique to surfaces in point cloud or implicit form.

2.3.1 Fast Marching on Cartesian grids

For simplicity, take the case of an interface propagating isotropically across a 2D rectangular orthogonal grid with position-dependent speed, or weight, function $F(x, y)$ away from a source (boundary) point (u, v). We are interested in the time of arrival, or offset distance, $T(x, y)$, of the front at grid point (x, y), i.e. the F-weighted Euclidean distance map of (u, v) is modelled by the arrival time function $T(x, y)$ given source (u, v). The relationship between the magnitude of the distance map's gradient, $\nabla T(x, y)$, and the given weight $F(x, y)$ can be expressed as the following boundary value formulation

$$\|\nabla T(x, y)\| = F(x, y), \tag{2.1}$$

with boundary condition $T(u, v) = 0$. That is, the norm of the distance map gradient is proportional to the weight function. The problem of determining a weighted intrinsic distance map[2] has therefore been transformed into the problem of solving a particular type of partial differential equation, the non-linear, so-called Eikonal equation. For $F(x, y)$ being strictly positive throughout, this type of equation can be solved for $T(x, y)$ in a very efficient manner using the original Fast Marching level set method as independently

[2]Note that for the planar domain case considered here, the weighted Euclidean and weighted intrinsic distance map coincide.

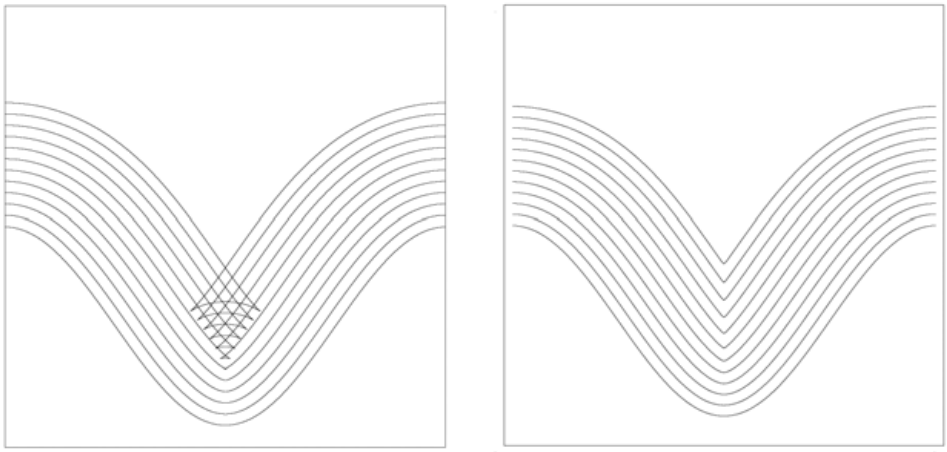

Figure 2.1: When propagating this cosine curve with unit speed, a corner develops which leads to the development of "swallowtails" when continuing the propagation (*left*). The arrival time of the front thus becomes multi-valued and no clear interface exists anymore. When observing the entropy condition that the arrival time computed at each underlying grid vertex is determined only once, this effect is avoided and the swallowtails disappear (*right*). The figure is adapted from [145].

introduced by Tsitsiklis [158], Helmsen et al. [70] and Sethian [144].

Since the Eikonal equation is well-known to become non-differentiable through the development of corners and cusps during propagation (Figure 2.1), the idea of the original Fast Marching method is to consider only *entropy-satisfying* finite difference approximations to the gradient for computing arrival time estimates. Entropy-satisfying here means that the particular finite difference approximation used prevents the arrival time for a grid vertex from being modified once it has been computed. That is, no new information is generated following the computation of a vertex' T-value. This way the envelope of the wave fronts is guaranteed to consist of first arrival times only. Although meeting this entropy condition still allows for the development of corners and cusps and thus local non-differentiability of the solution, the tails of the "swallowtails" generated during propagation are removed and the correct distance map values are obtained (Figure 2.1). As an example for a corresponding first-order (upwind)[3] approximation to the gradient operator in equation (2.1), consider [136]

$$\|\nabla T_{ij}\| \approx [max(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0)^2 + max(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0)^2]^{\frac{1}{2}} = F_{ij}, \tag{2.2}$$

where $F_{ij} \equiv F(i\Delta x, j\Delta y)$; $D_{ij}^{-x}T \equiv \frac{T_{ij}-T_{i-1j}}{\Delta x}$ and $D_{ij}^{+x}T \equiv \frac{T_{i+1j}-T_{ij}}{\Delta x}$ are the standard backward and forward derivative approximations with $\Delta x, \Delta y$ representing the (not nec-

[3] "Upwind" refers to the direction opposite to the direction of propagation, i.e. in direction of known arrival times.

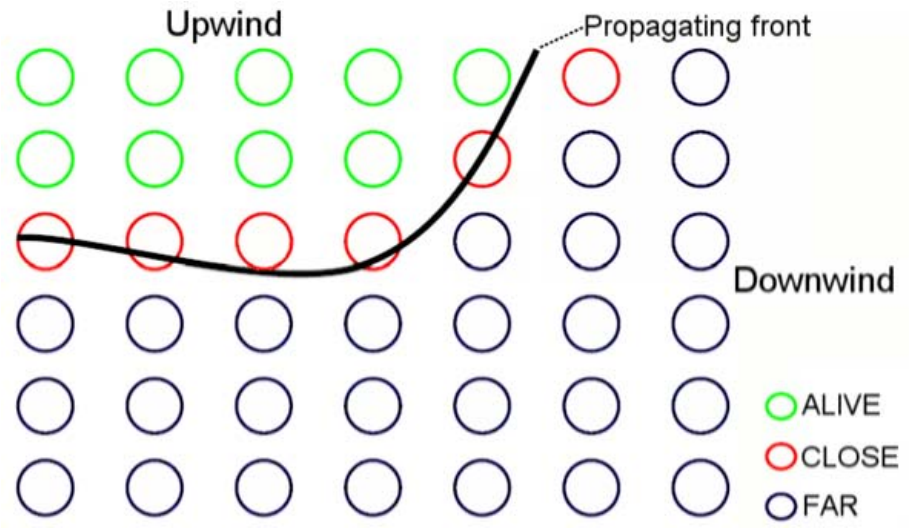

Figure 2.2: Fast Marching exploits the causality relationship of a particular upwind finite difference scheme in the form of a single-pass, narrow band algorithm operating in the downwind direction. As shown here for the 2D case, at any one step, the front is given by the narrow band of CLOSE grid vertices. Fast Marching updates their trial arrival times using previously computed, known T-values of neighbouring upwind (ALIVE) vertices. This way the front is sequentially moved outwards until all grid vertices have been visited.

essarily equal) grid spacing in the x- and y-directions respectively; equivalently for $D_{ij}^{-y}T$ and $D_{ij}^{+y}T$. T_{ij} is the discrete approximation to $T(i\Delta x, j\Delta y)$ on a Cartesian grid. The first-order nature of (2.2) stems from the use of a one-sided differencing scheme and means that the size of the approximation error is roughly proportional to Δx and Δy respectively [91].

The fact that this quadratic upwind difference approximation implies that information propagates from smaller to larger values of T only, i.e. a grid point's arrival time gets updated by neighbouring (upwind) points with smaller T-values only, represents the key observation for the efficiency of Fast Marching. This *causality*, or *monotonicity property*, allows for the solution of (2.2) sequentially, one-by-one in the order of increasing T-values, i.e. in a "marching" fashion, rather than as a simultaneous system as would otherwise be the case [147]. This "decoupling" of the simultaneous system can be exploited in the form of a narrow band of candidate points around the front representing its outward (downwind) motion and establishing the ascending order of grid vertices. More specifically, the T-values of already visited (ALIVE) grid points are frozen. The neighbouring (CLOSE) vertices of ALIVE points are then inserted into the narrow band thereby marching the band forward and constructing the solution sequentially [145] (Figure 2.2). This basic Fast Marching algorithm can thus be summarised as shown in Algorithm 1. For ease of exposition, ALIVE, CLOSE and FAR are treated as sets in Algorithm 1. ALIVE and FAR

Input: Boundary (propagation source) point $q \in M$. Speed function $F > 0$. Grid spacing in each grid direction.
Output: Weighted geodesic distance map of q.

---

0 *** *Initialisation* ***
1 Insert q in ALIVE with arrival time 0;
2 Insert in CLOSE, all grid points neighbouring q;
3 Initialise the points in CLOSE using a gradient approximation such as (2.2);
4 Insert all other grid points in FAR with initial arrival times of "∞";
5
6 *** *Front propagation* ***
7 **REPEAT**
8 Let TRIAL denote the point in CLOSE featuring the smallest arrival time;
Remove TRIAL from CLOSE and insert it in ALIVE;
9 Move all neighbours of TRIAL which are FAR to CLOSE;
10 Using a gradient approximation such as (2.2), update the T-values of all CLOSE neighbours of TRIAL using only ALIVE points in the computation;
11 **UNTIL** all grid points are ALIVE;

Alg. 1: Cartesian Fast Marching algorithm in pseudocode.

should be considered as states of a grid vertex represented in the form of simple labels rather than points held in separate data structures. Only the members of the narrow band CLOSE are both labelled as such *and* held in a separate data structure, typically a min-heap [142]. Implementation aspects are discussed in detail in Section 2.5.1.

By sorting the vertices in CLOSE by their estimated arrival time in a min-heap, detection of the vertex featuring the smallest arrival time is limited to the constant-time extraction of the min-heap root. Thus, the complexity of the algorithm follows from the worst-case complexity of min-heap re-heapification after root extraction (line 8), vertex insertion (line 9) or T-value updating (line 10). This is $O(\log W)$, with W representing the number of heap elements. Since N steps are required to visit each of the N grid points, the complexity of the conventional Fast Marching technique is $O(N \log N)$. A single min-heap structure may be used in this context to simultaneously track multiple propagation fronts originating from different points in the domain, i.e. the algorithm can handle multiple source points simultaneously. Figure 2.3 presents examples of multiple unit-speed fronts propagated across a planar Cartesian grid using conventional Fast Marching.

Although the algorithm was presented in the context of a planar Cartesian grid, it extends in the natural way to higher-dimensional Cartesian grids by allowing for the extra dimensions in the gradient approximation (2.2). In the case of processing in three dimensions, for example, 6- instead of 4-connectivity neighbourhoods are to be considered and (2.2)

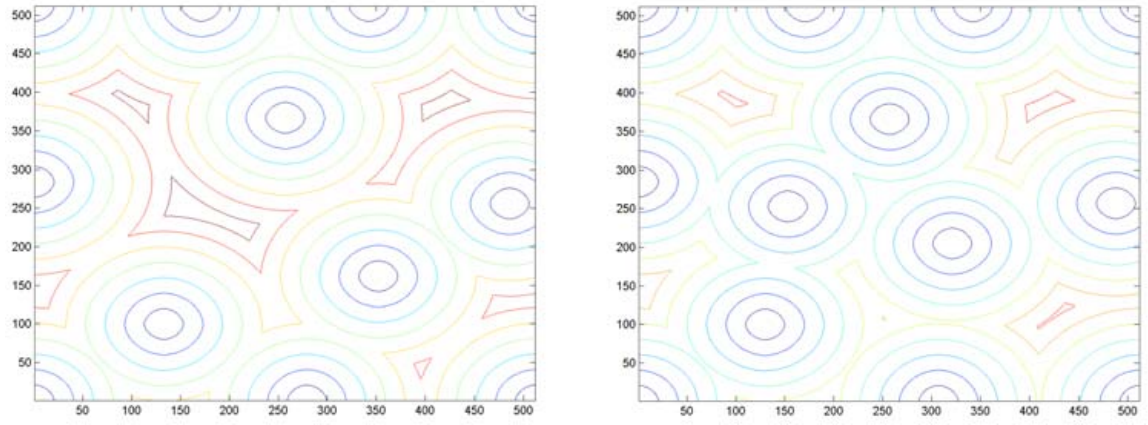

Figure 2.3: Equal distance contours of 12 (*left*) and 13 (*right*) propagation fronts computed across a (512×512) planar Cartesian grid using conventional Fast Marching.

may simply be extended to

$$\begin{aligned} \|\nabla T_{ijk}\| \approx \quad & [max(D_{ijk}^{-x}T, -D_{ijk}^{+x}T, 0)^2 \quad + \\ & max(D_{ijk}^{-y}T, -D_{ijk}^{+y}T, 0)^2 \quad + \\ & max(D_{ijk}^{-z}T, -D_{ijk}^{+z}T, 0)^2]^{\frac{1}{2}} \quad = F_{ijk}, \end{aligned} \tag{2.3}$$

with operators defined analogous to (2.2) above. By exploiting monotonicity, this scheme again allows for a very efficient solution analogous to Algorithm 1.

The Fast Marching technique is used in this thesis for distance mapping purposes due to a number of strengths. Firstly, unlike other front propagation algorithms (see, e.g. Cuisenaire [37] for a survey), Fast Marching is a single-pass technique, i.e. each grid point is touched only once, namely when it is assigned its final arrival time value. Furthermore, distance maps are computed with "sub-pixel" accuracy, the degree of which varies with the order of the approximation scheme and the grid resolution. Also, the geometry of the problem is taken into account by the gradient approximation of the arrival time function. This is in contrast to graph-based shortest path-type algorithms such as Dijkstra [42] whose consistency is undermined by metrication error caused by the restriction of paths to follow graph edges thereby addressing the distance mapping problem without attention to its geometry. As illustrated in Figure 2.4, the solution therefore generally does not converge to the exact solution as the mesh is refined. For a more detailed discussion of this issue, see Mitchell [108]. Finally, since the arrival time information is only propagated in the direction of increasing distance, the size of the narrow band remains small. In practice, the algorithm's complexity therefore tends to be closer to the theoretical optimum of $O(N)$ rather than $O(N \log N)$ [145].

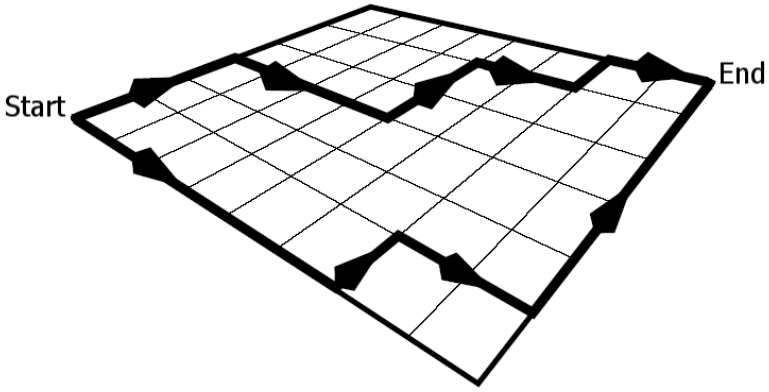

Figure 2.4: Due to the artificial metric introduced by the graph structure, graph-based distance mapping algorithms such as Dijkstra [42] suffer from metrication error. For the example illustrated here, the two paths are restricted to follow horizontal and vertical directions and, as a result, are of equal length. The true shortest distance between Start and End cannot be determined this way. This figure is adapted from [176].

As discussed in the following section, the Fast Marching principle can be extended to triangulated domains by devising a suitable upwind scheme for the approximation of the gradient operator across unstructured grids.

2.3.2 Fast Marching for triangulated surfaces

The conventional Fast Marching algorithm is extended to manifolds represented in triangular mesh form in Kimmel and Sethian [84]. In this context, a triangular mesh is a collection of vertices in $\mathbb{R}^m$ and a collection of triangles consisting of those vertices.

The underlying principle of Fast Marching for triangulated surfaces of devising an upwind finite difference scheme and exploiting its causality property in the form of a very efficient narrow band algorithm is identical to that of conventional Fast Marching discussed in detail in the previous section. The presentation is therefore restricted to those aspects which are substantially different from Fast Marching on structured grids. For more detailed treatments, see Novotni and Klein [118], Reimers [133] and Sethian and Vladimirsky [146].

Fast Marching for triangulated surfaces no longer considers points in a Cartesian grid but vertices of a triangular mesh. Front propagation occurs directly on this surface representation with given propagation speed $F(q_i)$ at triangle vertex q_i (Figure 2.5). Since we are dealing with unstructured grids now, the key point is that gradient approximations such as (2.2) are generally no longer applicable and a suitable upwind finite difference approximation for the extension of the marching principle to general triangulations needs to be

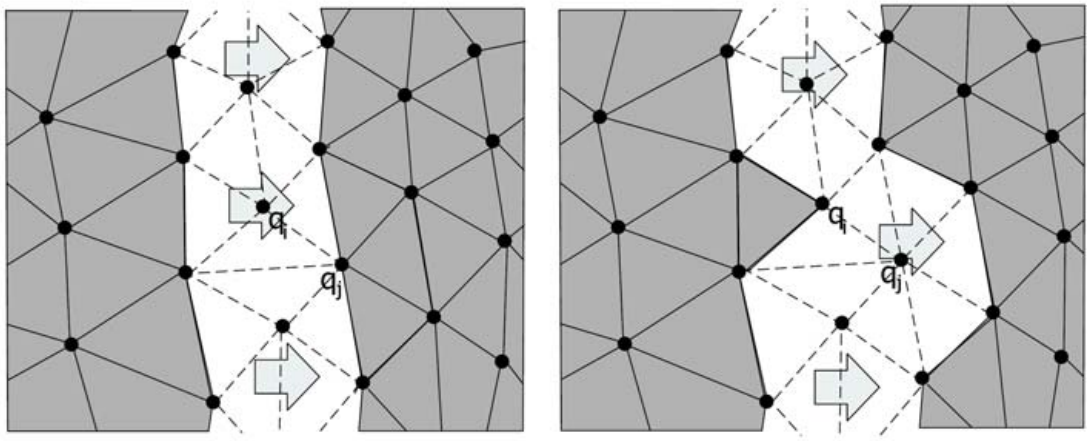

Figure 2.5: Fast Marching for triangulated surfaces propagates a narrow band of candidate vertices (white strip) from a source vertex outwards across a triangular mesh. The arrows indicate the downwind direction of the propagation (*left*). During the processing step depicted here, narrow band element q_i is found to have the smallest arrival time estimate and is thus included in the upwind set of ALIVE vertices. As a consequence, the vertex q_j is moved from the downwind set of FAR vertices to the set of CLOSE vertices, i.e. the narrow band, which is thereby moved outwards (*right*). Subsequent processing steps will include the other edge-adjacent neighbours of p_i into the narrow band. This figure is adapted from [118].

used instead. Once such a scheme is available, its causality property can be exploited again and the algorithmic approach of conventional Fast Marching carries over.

Kimmel and Sethian [84] develop a quadratic upwind gradient approximation for unstructured grids and based on this scheme, devise an update procedure for *acute triangulations*. Alternative upwind approximation schemes for triangulated surfaces have been developed by Barth and Sethian [13]. Kimmel and Sethian's [84] restriction to acute triangulations follows from the fact that in the case of acute triangles, the characteristic curve of a first-order PDE passing through a vertex q_i lies inside the triangle. More specifically, in the case of the first-order PDE (2.1), i.e. the Eikonal equation, its characteristic curve for q_i coincides with the gradient of the entropy-satisfying solution for $T(q_i)$ [147]. Thus, in the case of acute triangles, the gradient at q_i always points into the triangle from which it was updated (Figure 2.6). As a consequence, $T(q_i)$, firstly, only depends on the two other vertices of that triangle rather than the entire neighbourhood of q_i. Secondly, $T(q_i)$ is guaranteed to be at least as large as the maximum arrival time of the two other vertices, i.e. the arrival time function grows monotonically [147]. This causality property thus allows for proper upwinding as in the case of Cartesian Fast Marching and is shown by Tsitsiklis [158] to hold irrespective of the size of the (acute) triangle. As in the case of conventional Fast Marching, this property can be exploited for the design of a very efficient single-pass updating scheme which, for each triangle sharing the vertex under

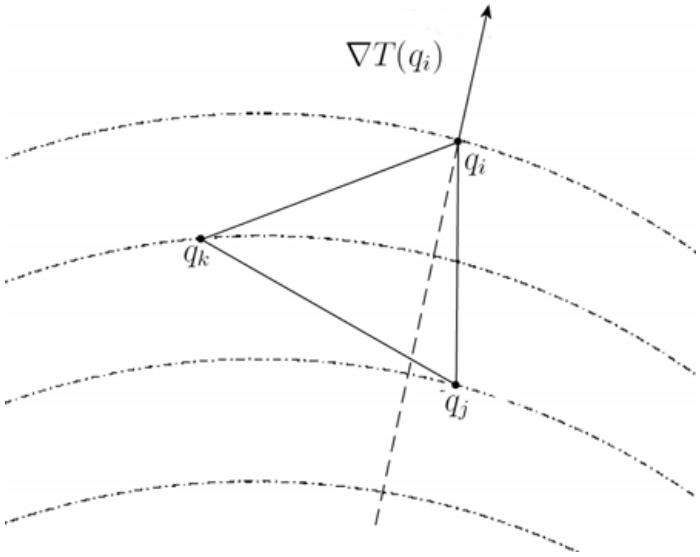

Figure 2.6: In the case of an acute triangle, the characteristic curve of a first-order PDE such as the Eikonal equation passing through a vertex q_i coincides with the gradient of the entropy-satisfying solution for $T(q_i)$, i.e. in the case of acute triangles the gradient always points into the triangle from which it was updated and the monotonicity/causality property required for Fast Marching is given. In the case depicted here, the already ALIVE vertices q_j and q_k can be used to support the computation of the arrival time at q_i. This figure is adapted from [147].

consideration, only needs to take into account the known (ALIVE) T-values at the other two triangle vertices rather than the T-values at all the neighbouring vertices. Thus, Kimmel and Sethian's [84] algorithm computes arrival time estimates for a vertex q_i from the two ALIVE vertices previously passed by the front and belonging to any of the triangles sharing q_i (Figure 2.6).

Consider a triangle $\triangle ABC$ with two ALIVE vertices A, B and associated arrival times $T(A), T(B)$, $T(B) > T(A)$. The problem then is to devise an update scheme for the T-value at C. To solve this problem, the basic idea is to determine a value $t = T(C) - T(A)$ at C such that the plane over $\triangle ABC$ determined by $T(A), T(B)$ and $T(C)$ has a gradient equal to the given weight $F(C)$. This way the update scheme will take into account the geometry of the problem. As illustrated in Figure 2.7, t, i.e. the distance by which the plane is to be raised at C, follows as $(t - u)/h = F(C)$, with $u = T(B) - T(A)$ and h denoting the altitude of $\triangle CDB$. Using trigonometric arguments, Kimmel and Sethian [84] derive from this expression the following first-order quadratic upwind scheme for t

$$(a^2 + b^2 - 2ab\cos\theta)t^2 + 2bu(a\cos\theta - b)t + b^2(u^2 - F(C)^2a^2\sin^2\theta) = 0 \qquad (2.4)$$

This scheme only applies if the monotonicity condition of $T(C)$ being updated from within $\triangle ABC$ holds true, i.e. if sufficient numerical support in the form of two ALIVE vertices

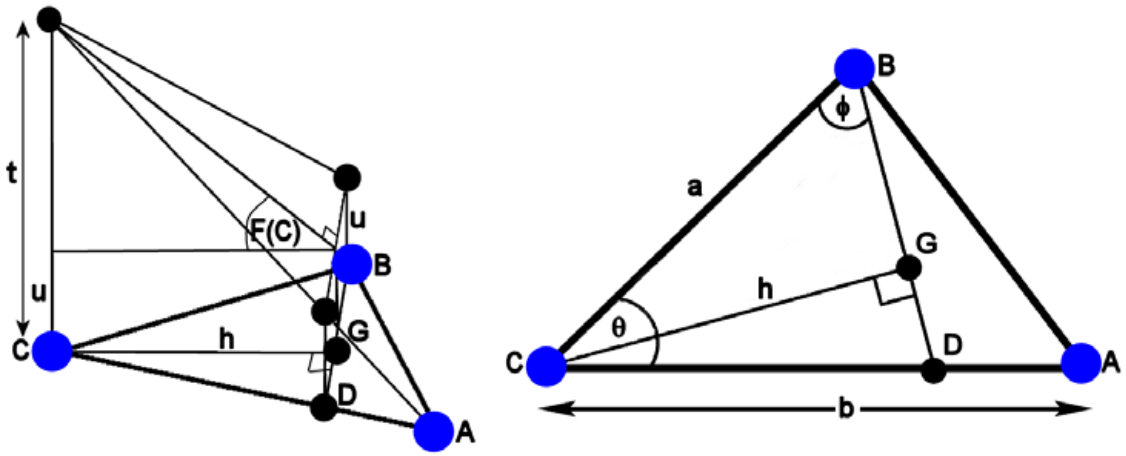

Figure 2.7: Distance value t at vertex C in triangle ABC is found by constructing a plane over ABC with gradient $F(C)$. That is, $t = T(C) - T(A)$ is sought such that $(t-u)/h = F(C)$ (*left*). This expression leads to a quadratic upwind scheme in t. The trigonometric relationships used in the derivation of both the scheme and its monotonicity condition are illustrated on the right. This figure is adopted from [84].

is given. For this to be the case, edge CD needs to meet the following condition

$$CD = \frac{b(T(C)-u)}{t} \in \left[a\cos\theta; \frac{a}{\cos\theta}\right], \tag{2.5}$$

This follows from $\angle BDC = \pi/2$, if $G = D$, and $\angle CBD = \pi/2$, if $G = B$, so that CD should be limited by $CD = a\cos\theta$ in the former and $CD = \frac{a}{\cos\theta}$ in the latter case. Thus, (2.5) ensures that point G will be located on the line between D and B, i.e. the triangle altitude h is ensured to lie within $\triangle CDB$, and the update step is guaranteed to be performed from within $\triangle ABC$. Thus, if (2.5) holds true, the update scheme (2.4) can be used to determine a value for t and $T(C)$ is estimated as $T(C) = \min\{T(C), T(A)+t\}$; otherwise $T(C)$ follows as $T(C) = \min\{T(C), T(A) + bF(C), T(B) + aF(C)\}$.

Condition (2.5) being true no longer ensures that causality holds when dealing with obtuse triangles. For the case of general triangulations, Kimmel and Sethian [84] therefore suggest to split any obtuse triangles into acute triangles during a pre-processing step to generate sufficient numerical support for the updating step. Note that this may require the recursive unfolding of triangles adjacent to the obtuse triangle under consideration until a new vertex is found, the "virtual" edge to which splits the obtuse into two acute angles [84]. That is, the solution is effectively allowed to depend on vertices which are not connected to the vertex under consideration in order to enforce the causality principle. I opt for a simpler alternative approach towards dealing with obtuse triangles given alongside other implementation details in Section 3.7.1.

Input: Acute triangulation of P. Boundary (propagation source) vertex $q \in P$. Speed function $F > 0$.
Output: Weighted geodesic distance map of q.

---

0 *** *Initialisation* ***
1 Insert q in ALIVE with arrival time 0;
2 Insert in CLOSE, all triangle vertices edge-adjacent to q;
3 Initialise the points in CLOSE using a gradient approximation such as (2.4);
4 Insert all other triangle vertices in FAR with initial arrival times of "∞";
5
6 *** *Front propagation* ***
7 **REPEAT**
8 Let TRIAL denote the vertex in CLOSE featuring the smallest arrival time; Remove TRIAL from CLOSE and insert it in ALIVE;
9 Move all edge-adjacent neighbours of TRIAL which are FAR to CLOSE;
10 Using a gradient approximation such as (2.4), update the T-values of all CLOSE neighbours of TRIAL using only ALIVE vertices in the computation;
11 **UNTIL** all vertices are ALIVE;

Alg. 2: Fast Marching algorithm for triangulated surfaces in pseudocode.

As in the case of conventional Fast Marching, Kimmel and Sethian's [84] method then selects the smallest estimate as the final approximation of $T(C)$ from the possibly relatively large number of estimates computed from the triangles sharing vertex C. The resulting algorithm is summarised in Algorithm 2.

The following section deals with the extension of the Fast Marching principle to surfaces in point cloud and implicit form.

2.3.3 Fast Marching for surfaces in point cloud and implicit form

Mémoli and Sapiro [103, 104] extend the applicability of the conventional Fast Marching idea to the case of general co-dimension manifolds in point cloud and implicit form respectively in three or higher dimensions. In the following, these important extensions are discussed in detail.

Take the case of a surface given in point cloud form and consider the constant radius r-offset Ω_P^r, i.e. the union of Euclidean balls with radius r centred at points $p_i \in P$ (Figure 2.8)

$$\Omega_P^r := \bigcup_{i=1}^{\|P\|} B(p_i, r) = \{x \in \mathbb{R}^m : d(p_i, x) \leq r\}$$

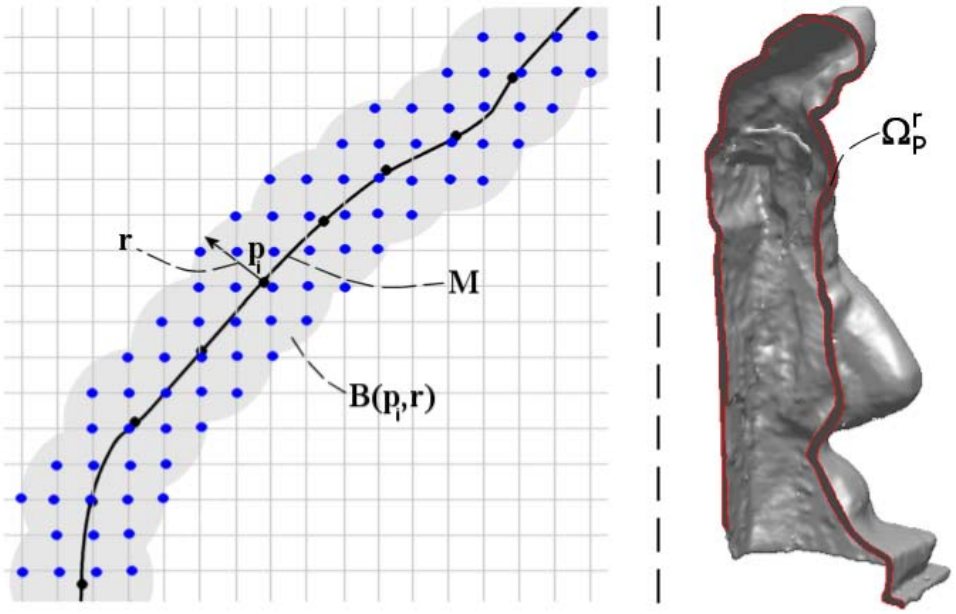

Figure 2.8: Fast Marching for surfaces in point cloud form [104] operates in an (not necessarily constant radius) offset band consisting of the union of balls $B(p_i, r)$ centred at (black) points p_i of the surface M (*left*). Only those (blue) grid points falling inside the offset band are considered during processing. A cross-sectional view (in the form of a planar cut) of a constant radius offset band for the Michelangelo Youthful data set is shown on the right.

To approximate the weighted intrinsic distance map, T_M on M, originating from a source point $q \in M$, Mémoli and Sapiro [104] suggest computing the Euclidean distance map in Ω_P^r, denoted $T_{\Omega_P^r}$. That is

$$\|\nabla_M T_M(p)\| = F(p), \tag{2.6}$$

for $p \in M$ and with boundary condition $T_M(q) = 0$ is approximated by

$$\|\nabla T_{\Omega_P^r}(p)\| = \tilde{F}(p), \tag{2.7}$$

for $p \in \Omega_P^r$ and boundary condition $T_{\Omega_P^r}(q) = 0$. $\tilde{F}$ represents the (smooth) extension of the propagation speed F on M into Ω_P^r;[4] ∇_M denotes the intrinsic vector differential operator. The problem of computing a weighted intrinsic distance map is therefore transformed into the problem of computing a weighted Euclidean, or extrinsic, distance map in the offset band Ω_P^r around the surface, i.e. in an Euclidean manifold with boundary.

Mémoli and Sapiro [104] prove uniform, probabilistic convergence between these two dis-

[4]For non-uniform speed functions, the question arises of how to compute $\tilde{F}$. If the normals of M are given or can be estimated, F may be smoothly extended into Ω_P^r by orthogonal projection, i.e. $\tilde{F}(p) = F\left(\Pi_M(p)\right), p \in \Omega_P^r$, where $\Pi_M(\cdot)$ denotes the orthogonal projection operator from $\mathbb{R}^m$ onto M. For more detail, see Mémoli and Sapiro [103].

Input: Point cloud P. Offset band Ω_P^r. Boundary (propagation source) point $q \in \Omega_P^r$. Speed function $\tilde{F} > 0$. Grid spacing in each grid direction.
Output: Weighted geodesic distance map of q.

```
*** Initialisation ***
    Insert q in ALIVE with arrival time 0;
    Insert in CLOSE, all grid points neighbouring q and which fall inside Ω_P^r;
    Initialise the points in CLOSE using a gradient approximation such as (2.3);
    Insert all other grid points in Ω_P^r in FAR with initial arrival times of "∞";

*** Front propagation ***
    REPEAT
        Let TRIAL denote the point in CLOSE featuring the smallest arrival time;
        Remove TRIAL from CLOSE and insert it in ALIVE;
        Move all FAR neighbours of TRIAL which belong to Ω_P^r to CLOSE;
        Using a gradient approximation such as (2.3), update the T-values of
        all CLOSE neighbours of TRIAL using only ALIVE points in the computation;
    UNTIL all grid points in Ω_P^r are ALIVE;
```

Alg. 3: Fast Marching algorithm for point clouds in pseudocode.

tance maps, and geodesics computed from them, *for both noise-free and noisy* (provided noise is bounded from above by r), randomly-sampled point clouds and thus show that the approximation error between the intrinsic and extrinsic distance maps is of the same theoretical order as that of the conventional Fast Marching algorithm [145]. With the order of the numerical approximation remaining unchanged, Fast Marching can be used to approximate the solution to (2.7) in a computationally optimal manner and without the need for any prior surface reconstruction by only slightly modifying the conventional Cartesian Fast Marching technique to deal with bounded spaces as summarised in Algorithm 3. This is achieved by simply restricting the grid points visited by the conventional Fast Marching algorithm to those located in Ω_P^r. By performing the computations within this offset band, this method is relatively robust in the presence of noisy point samples, especially when compared to graph-based distance mapping algorithms such as Giesen and Wagner [54] and Tenenbaum et al. [156] in which case the geodesics pass through the noisy samples rather than an union of Euclidean balls centred at the input points.

When again using a min-heap for the optimal ordering of narrow band members, the complexity of this algorithm is $O(N \log N)$, where N represents the number of grid points located in Ω_P^r [104]. Memory efficiency is achieved by storing these grid points only as opposed to the entire discretised bounding volume. Note in this context that subject to the bounds given in Mémoli and Sapiro [104], r will generally be small and does not have to be constant but may vary with each p_i so that the offset band will usually consist of only a fraction of the grid vertices making up the bounding volume. Details of my

implementation of this method are discussed in Section 2.5.2.

Although this Fast Marching technique has been introduced in the context of a surface represented in point cloud form, the concept carries over analogously to implicit surfaces. In the implicit surface case, M is represented by a closed hyper-surface in $\mathbb{R}^m$ given as the zero level set of a distance function $\phi : \mathbb{R}^m \rightarrow \mathbb{R}$. The offset band of radius r, Ω_r, now follows as $\Omega_r := \bigcup_{x \in M} B(x, r) = \{x \in \mathbb{R}^m : \|\phi(x)\| \leq r\}$. Mémoli and Sapiro [103] subsequently show uniform (deterministic) convergence between the weighted extrinsic distance map computed in Ω_r and its intrinsic counterpart. Since this thesis is predominantly concerned with the processing of surfaces in point cloud form, the reader is referred to Mémoli and Sapiro [103] for more detail.

I use the Fast Marching techniques discussed above for the support of, amongst other things, intrinsic meshless surface subdivision and the intrinsic point sampling of surfaces in point cloud form without the need for any prior or intermediate surface reconstruction. As part of these applications, mesh connectivity is replaced by proximity information for the estimation of surface properties. In the following, existing neighbourhood concepts for point-sampled geometry are therefore reviewed. I address limitations of these exclusively extrinsic concepts by proposing an alternative intrinsic neighbourhood concept.

2.4 Neighbourhood concepts for point-sampled geometry

Many advantages of point-sampled geometry over mesh-based representations follow from the absence of any connectivity information and the associated overhead. However, when dealing with point-sampled geometry in its most abstract form of representation, i.e. positions in mD space without normal information, tasks such as local surface approximation (Section 4.3.1) and point set re- or upsampling (Section 4.3 and 5.2.1 respectively) require proximity information to compute differential properties such as surface normals or local curvature. In Section 2.4.1, I review some of the pitfalls associated with the collection of proximity information for point-sampled geometry. The most widely-used neighbourhood concepts are then discussed in Section 2.4.2 with respect to these pitfalls. Section 2.4.3 presents my alternative, intrinsic neighbourhood concept.

2.4.1 Pitfalls for point-based neighbourhood concepts

Whilst in the case of mesh-based surface representations, neighbourhood information is already given in the form of mesh connectivity, point-based processing needs to collect this information. Depending on the topology of the underlying, unknown surface, this can represent a difficult task. The most frequently encountered problems include (Figure 2.9)

---

a) Assignment of points from disjoint sheets of the surface to the same neighbourhood.
b) Skewed neighbour distributions near local point density non-uniformities.
c) Overlapping neighbour relations.

The occurrence of problem (a) implies that the proximity information cannot be used for meaningful further processing since it connects disjoint parts of the underlying surface. In the case of problem (b), neighbours are not distributed all around the point under consideration. Gaps in the point distribution therefore tend not to be covered by such neighbourhoods and their use for the support of point set re- or upsampling would aggravate the existing irregularities in the form of local clustering. Even if dealing with uniformly distributed data, local non-uniformities are introduced when using overlapping neighbour relations to support point set upsampling. This is illustrated in Figure 2.9(c). For the example shown, the neighbourhoods' independent determination has led to p_m being associated with the neighbourhoods of both p_l and p_k. As a result, the neighbour relation of p_m with p_k overlaps with the relation of p_n with p_l. When this neighbourhood information is used for, for example, the support of upsampling by inserting new points "midedge" between neighbours, this overlap would introduce non-uniformities in the form of local clustering near the intersection of the two red "edges".

To be useful for the re- and upsampling purposes of this thesis, a neighbourhood concept should avoid any occurrence of problems (a)-(c). Note that these problems are closely related to the fact that surface points which are close in Euclidean distance can be far away from each other in intrinsic distance. Thus, to obtain good estimates of the differential properties at a point, proximity should preferably be measured intrinsically rather than extrinsically. Following the discussion of the most widely-used (Euclidean) neighbourhood concepts, a corresponding intrinsic neighbourhood definition is presented in Section 2.4.3.

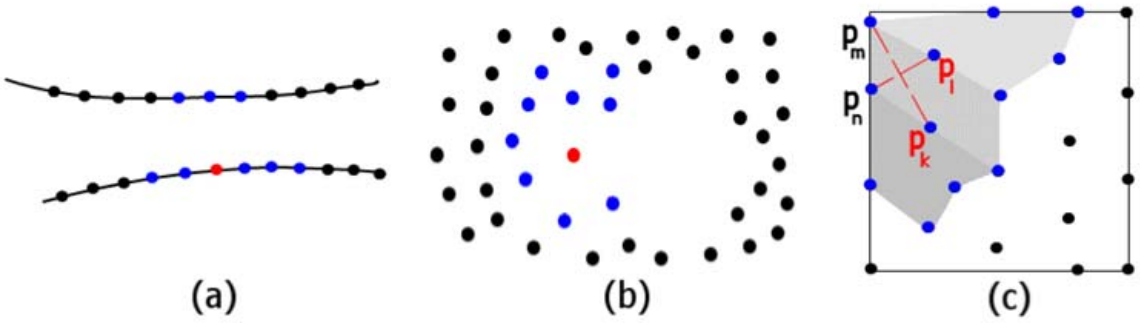

Figure 2.9: Typical pitfalls for point-based neighbourhood concepts include the assignment of points from disjoint sheets of the surface to the (blue) neighbourhood of an (red) input point (a), the skewed distribution of neighbours at the presence of local point density non-uniformities (b) and the overlap of (red-lined) neighbour relations due to the independent determination of the point neighbourhoods (c). For the overlapping relations example shown here, the neighbourhood for point p_k (here: 8 nearest neighbourhoods) is shown in dark grey, the corresponding neighbourhood of point p_l is shown in light grey. The patterned region covers those (blue) neighbours included in both neighbourhoods. The red "edges" denote the resulting neighbour relations amongst the points p_k and p_m on the one and p_l and p_n on the other hand.

2.4.2 Conventional extrinsic neighbourhood concepts

The most widely-used neighbourhood concepts for point-sampled geometry are defined in the Euclidean distance or make use of extrinsic combinatorial data structures fitted to the point set. These neighbourhood concepts consist of, in increasing order of usefulness, Euclidean ball neighbourhoods, k nearest and restricted Delaunay neighbourhoods and are reviewed in the following with respect to problems (a)-(c). Note that the various concepts discussed in this section assume throughout that the given point set represents an adequate sampling of the underlying surface.

The *Euclidean ball neighbourhood* EBN_{p_i} of point $p_i \in P$ considers all points in P located within a sphere of radius r centred at p_i, i.e. $EBN_{p_i} = \{p_j \in P : d(p_i, p_j) \leq r\}$ (Figure 2.10(a)). For a value of r greater than the local feature size [8] (Appendix B), disjoint parts of the surface will be assigned to the same EBN_{p_i} making subsequent processing on the basis of this neighbourhood information invalid. Secondly, to avoid skewed neighbour distributions at the presence of local point distribution non-uniformities, the ball radii need to be made adaptive to account for the variation in point density. Any method for the estimation of radii r_i such as minimum spanning trees or other graph structures [143] tend to undermine the most important advantage of (constant radius) Euclidean ball neighbourhoods, their ease of computation. Finally, since Euclidean ball neighbourhoods are computed for each point independently, they are prone to produce overlapping neighbour relations. As a result, their applicability is generally limited to

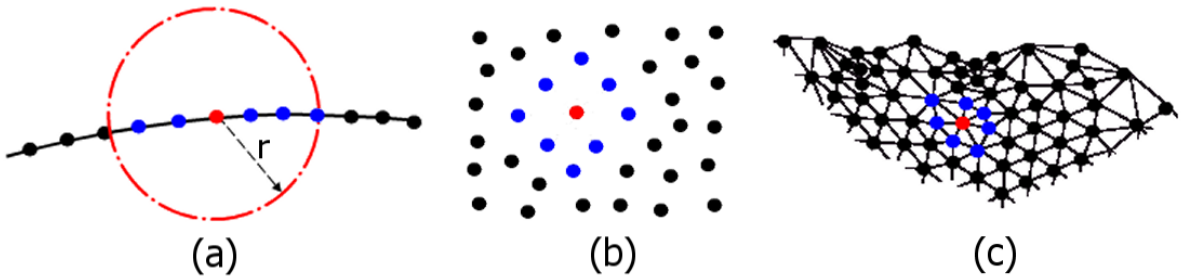

Figure 2.10: Euclidean ball neighbourhood of radius r (a), $k(= 8)$ nearest neighbourhood (b) and restricted Euclidean Delaunay neighbourhood (c) of a (red) input point. Neighbours are represented in blue.

very uniformly distributed points sets for which it is possible to set the constant radius of the Euclidean balls to a value consistently less than the local feature size.

The Euclidean k *nearest neighbourhood* NN_{p_i} of p_i consists of the k points in P closest in Euclidean distance to p_i (Figure 2.10(b)). Since NN_{p_i} is defined in terms of Euclidean distances, problem (a) is not guaranteed to be avoided. In addition, the question needs to be addressed what represents a useful value for k. Hoppe et al. [72] assume k to be given as an input parameter. Pauly et al. [125] select k experimentally for the purpose of local normal and surface variation estimation respectively. Mitra and Nguyen [109] suggest a theoretical approach whose practical value, however, suffers from the need for estimates of both the standard deviation of any noise affecting the point positions and the local sampling density as well as the local curvature. These estimates, which partly themselves require the use of local neighbourhoods, are to be re-computed at or around each point. Irrespective of whether or not these values are easily obtainable, the analysis applies to the case of locally regularly uniformly distributed point sets only. At the presence of point distribution non-uniformities, k nearest neighbourhoods tend to be skewed [97]. The *enhanced k nearest neighbourhood* idea [97] tries to overcome this particular problem by controlling for a maximum angle between pairs of neighbours sorted around the point under consideration (Figure 2.11). This concept is used in the context of point set resampling in Section 4.3. Note, however, that due to the fact that each point's enhanced k nearest neighbourhood is determined independently, this or the similar modification of the k nearest neighbourhood concept put forward by Guennebaud et al. [63] does not help to avoid the issue of overlapping neighbour relations [63].

The *restricted Delaunay neighbourhood* of p_i consists of its neighbours in the global restricted Euclidean Delaunay triangulation of the surface [20] (Figure 2.10(c)).[5] Since this neighbourhood concept is based on a subset of an extrinsic mesh data structure

[5]For a definition of the notion of restricted Euclidean Delaunay triangulations, see Appendix B.

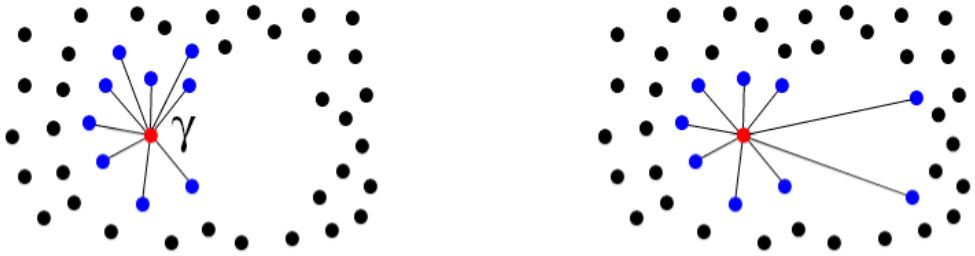

Figure 2.11: The conventional $k(= 9)$ nearest neighbourhood of a (red) point p_i (*left*) fails to account for the local non-uniformity in the point distribution and is skewed. This makes it unsuitable for the support of tasks such as point cloud re- or upsampling. The enhanced 9 nearest neighbourhood (*right*) avoids this effect by controlling for violations such as γ of an angular threshold between successive neighbours.

partitioning the underlying surface, point proximity follows from mesh connectivity and point neighbourhoods are not determined independently. Problems (b)-(c) are therefore avoided. The avoidance of problem (a) depends on how well the restricted Delaunay triangulation captures the topology of the underlying surface. Topological guarantees have been derived in this context by Amenta et al. [8,9] for point sets of sufficiently high density. See also Leibon and Letscher [90]. In the case of point sets not meeting this condition, the Delaunay triangulation may connect disjoint surface sheets so that problem (a) is no longer guaranteed to be prevented.

The generally favourable properties of the restricted Delaunay neighbourhood come at the expense of the frequently prohibitively costly computation of the Euclidean Delaunay triangulation in mD and its ("restricted") subset intersecting the underlying surface, as discussed by, for example, Boissonnat and Cazals [20]. As shown by Andersson et al. [12], in the case of uniformly distributed point sets, the restricted Delaunay neighbourhood of p_i may be approximated efficiently from a k nearest neighbourhood of p_i. However, apart from being inapplicable in the case of non-uniformly distributed point clouds, the restricted Delaunay neighbourhood is then approximated locally and independently rather than being derived from a global surface partitioning and the problem of overlapping neighbour relations is no longer avoided.

In the following section, I propose to derive proximity information intrinsically instead and put forward an intrinsic neighbourhood concept.

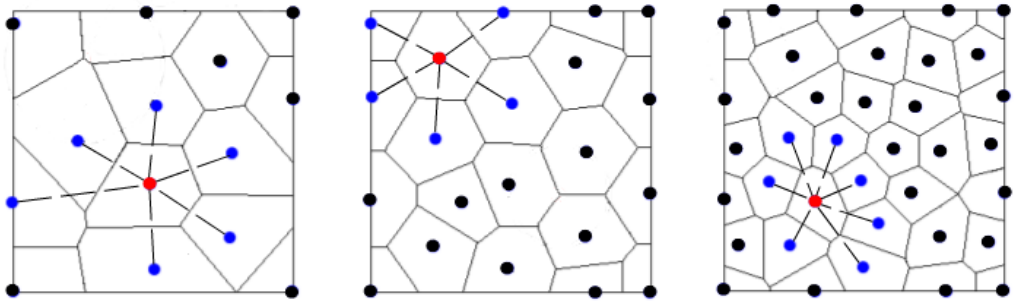

Figure 2.12: Examples for natural neighbourhoods (dashed lines) as defined by the Voronoi diagrams (solid lines) of planar input point sets. For this planar example, the Euclidean and intrinsic Voronoi diagrams and thus the Euclidean natural [151] and intrinsic natural neighbourhoods coincide. This does not extend to the more general, non-planar surface case.

2.4.3 An intrinsic neighbourhood concept

To determine point neighbourhoods intrinsically, I propose to use the set of neighbours of p_i in the geodesic Voronoi diagram of P, $VD(P)$, i.e. an *intrinsic* "natural" [151], or Voronoi, neighbourhood (Figure 2.12),

$$\mathcal{N}_{p_i} = \{p_j : p_i \text{ and } p_j \text{ are neighbours in } VD(P)\},$$

for p_i, $p_j \in P$, $p_i \neq p_j$; p_i and p_j are neighbours in $VD(P)$, if $R(p_i, P) \cap R(p_j, P)$ is neither empty nor a singleton, i.e. the relevant Voronoi regions share an edge. Thus, p_i is a Voronoi neighbour of p_j if and only if p_j is a Voronoi neighbour of p_i.

As regards problems (a)-(c), this neighbourhood concept benefits from its definition in terms of a global surface partitioning. The consideration of intrinsic rather than Euclidean interpoint distances for this partitioning avoids points from disjoint surface sheets to be assigned to the same neighbourhood. Its global nature means that intrinsic natural neighbourhoods are well-defined across areas of local non-uniformities in the point distribution. Finally, the use of a Voronoi partitioning in the definition of $\mathcal{N}_{p_i}$ implies that intrinsic natural neighbourhoods are not determined independently of each other but rather implicitly take into account proximity to all other points via their intrinsic Voronoi diagram. The problem of overlapping neighbour relations is thereby avoided (Figure 2.13).

The implementation of intrinsic natural neighbourhoods is discussed in the context of intrinsic meshless surface subdivision in Section 5.4.

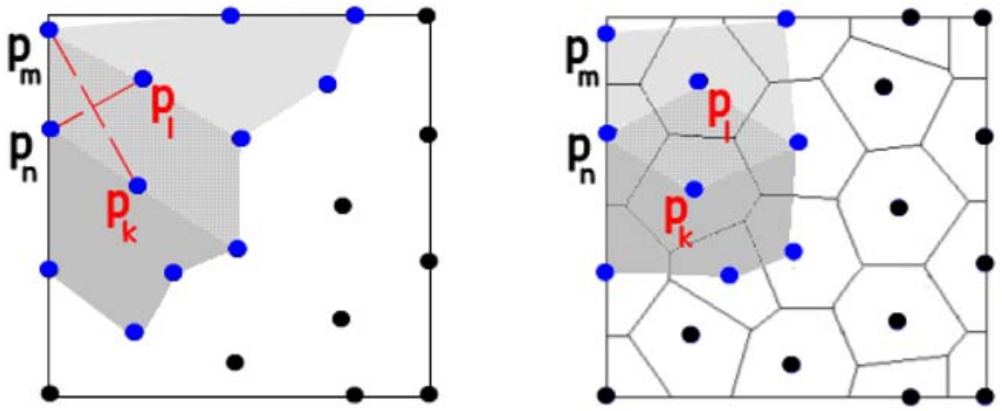

Figure 2.13: The overlapping relations example of Figure 2.9 is reproduced on the left. Due to the determination of the intrinsic natural neighbourhoods of p_k and p_l from a global Voronoi partitioning, no such neighbour relation is established between p_m and p_k and the problem is avoided when using intrinsic natural instead of k nearest neighbourhoods (*right*). Note that for this planar case, the intrinsic natural neighbourhoods represent a subset of the k nearest neighbourhoods. This does not apply in the more general, non-planar surface case.

2.5 Implementation details

As in the case of their algorithmic details, the implementations of conventional Cartesian Fast Marching and Fast Marching for surfaces in point cloud form overlap substantially. Also, the latter Fast Marching technique is at the heart of both the intrinsic meshless surface simplification algorithm of Chapter 4 and the intrinsic meshless surface subdivision algorithm of Chapter 5. These implementations are therefore described in Sections 2.5.1 and 2.5.2 below rather than the relevant work chapters. Similarly, geodesic Voronoi diagrams are used repeatedly in subsequent chapters. The corresponding implementation details are therefore summarised in Section 2.5.3.

2.5.1 Fast Marching on Cartesian grids

The implementation of the original Cartesian Fast Marching algorithm [70, 144, 158] requires two main data structures, a m-dimensional, not necessarily regular, structured grid and a min-heap. As illustrated by equation (2.2), any irregularity of the grid, i.e. any differences in grid spacing in the grid directions, is explicitly allowed for in the finite difference approximation. I opt for the implementation of the grid as a mapping from a unique key generated from the Cartesian coordinates of a grid vertex to a grid element object holding the vertex' arrival time, its state (FAR, CLOSE, ALIVE) and the indices of its heap entries. Figure 2.14 illustrates this design. Apart from the grid spacing flexibility

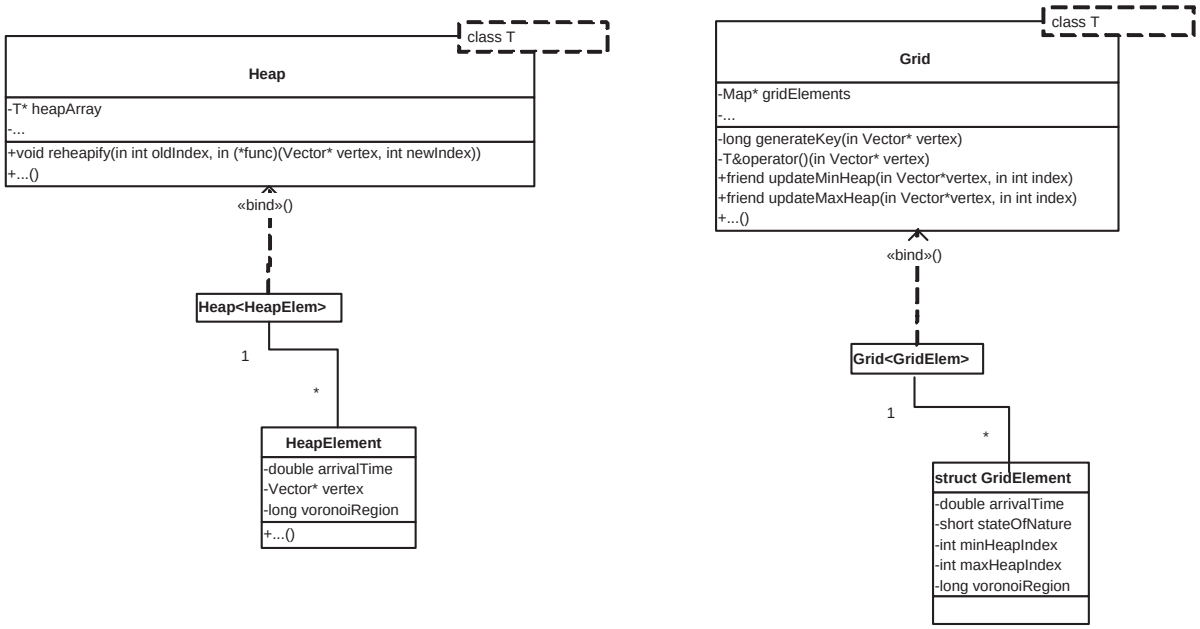

Figure 2.14: Extracts of my class design of the implementation of Cartesian Fast Marching in Unified Modeling Language (UML) [52] syntax.

of such a mapping, this design is beneficial in a memory efficiency sense since it allows to drop grid elements from consideration which are known not to be needed for further processing.

The min-heap is implemented as a templated class with the heap elements stored in an array. Thus, for each CLOSE vertex stored at array position $i \geq 1$, its children can be found in constant time at positions $2i$ and $2i+1$ respectively. Similarly, the parent of each element located at position $i > 1$ is found in constant time at position $\lfloor i/2 \rfloor$ [142]. Since the heap elements are keyed by their arrival time estimates, the root, i.e. $i = 1$, holds the minimal arrival time estimate at any time. Apart from this key, each heap element holds a pointer to the grid vertex whose arrival time estimate is being stored and the index of the input point whose Voronoi region it belongs to (Figure 2.14), as determined during front propagation (Section 5.4.1).

Following root extraction, vertex insertion or T-estimate updating, the positions of heap elements will generally change following the enforcement of the heap property, i.e. re-heapification. In this case, the heap indices stored at those grid vertices whose heap entries have changed need to be updated accordingly. This lack of separation between information stored at grid vertices and information stored in the heap represents the only implementational complication of Fast Marching. This complication is resolved here by passing a function pointer to the heap's re-heapification method. That is, a pointer to a function is passed which takes a pointer to a grid vertex and its new heap index as

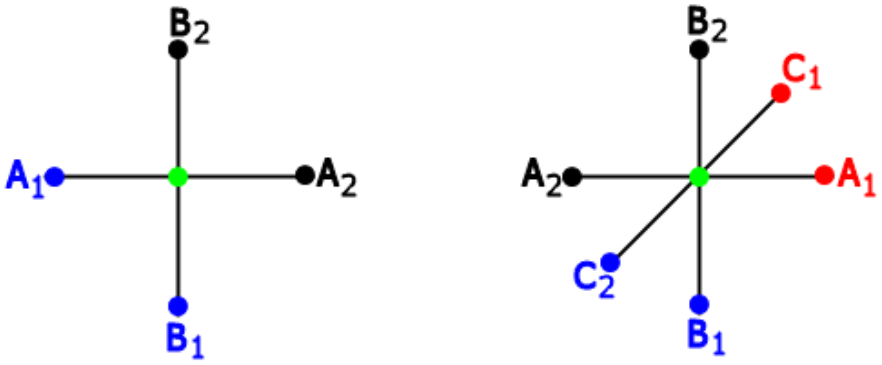

Figure 2.15: Only (the coloured) ALIVE grid vertices are considered for the solution of (2.3) at a (green) vertex. Given pairwise opposite ALIVE vertices, the vertex featuring the smaller T-value is included in the computation. In the 2D case shown here (*left*) no such pair of opposite ALIVE vertices exists. Of the two (blue) ALIVE vertices, the one featuring the smaller T-value is labelled A_1 so that $T_{A_1} \leq T_{B_1}$ holds. In the 3D example shown on the right, vertices C_1 and C_2 are both ALIVE but C_1 features a smaller arrival time, i.e. is located closer to the source point of the red front than C_2 to the source point of the blue front. Thus, the T-value at vertex C_1 is to be considered for $T_{C_1} \leq T_{C_2}$ to hold. The other vertices are selected such that the additional condition $T_{A_1} \leq T_{B_1} \leq T_{C_1}$ applies as well.

parameters and updates the heap index information stored at the grid vertex accordingly. This is implemented by allowing the function "friendly" access to the grid class (Figure 2.14). Although this solution adds the overhead of function pointer maintenance, it has the advantage of allowing me to use the same heap class not only for the instantiation of a min- but also a max-heap object used alongside the same grid object. To maintain the corresponding heap indices at a particular grid vertex, the function pointer-based implementation of the heap class remains unchanged. During processing, the heap objects get passed a pointer to one of the two friendly functions depending on whether the min- or max-heap index at a grid vertex needs to be updated.

Although this class design is presented in the context of Cartesian Fast Marching, the heap and heap element classes and the grid element structure are not specific to this implementation and are re-used for the implementations of Fast Marching for surfaces in point cloud and triangular mesh form. Similarly, for the implementation of Fast Marching for surfaces in point cloud form, the (structured) grid class can also be re-used.

With the main data structures in place, how to set up and solve the quadratic finite difference approximation represents the only remaining implementation question. Let us assume, without loss of generality, that we are dealing with a Cartesian Fast Marching problem in 3D, i.e. the quadratic finite difference approximation (2.3). At each updating step, the 6-connectivity neighbourhood of the vertex whose T_{ijk}-value is to be updated is

considered. Following Algorithm 1, only ALIVE vertices need to be taken into account in the computation of the T_{ijk}-estimate.

Using the notation of Deschamps and Cohen [40], let $\{A_1, A_2\}$, $\{B_1, B_2\}$ and $\{C_1, C_2\}$ represent the opposite pairs of neighbours in the 6-connectivity neighbourhood around vertex $(i\Delta x, j\Delta y, k\Delta z)$ whose T_{ijk}-value is to be updated. The couples are chosen such that $T_{A_1} \leq T_{A_2}$, $T_{B_1} \leq T_{B_2}$, $T_{C_1} \leq T_{C_2}$ and $T_{A_1} \leq T_{B_1} \leq T_{C_1}$ holds (Figure 2.15). The monotonicity property of Fast Marching requires that any solution meets the condition $T_{ijk} \geq T_{C_1}$. From the finite difference approximation (2.3), the quadratic equation to be solved is

$$(T_{ijk} - T_{C_1})^2 + (T_{ijk} - T_{B_1})^2 + (T_{ijk} - T_{A_1})^2 - F_{ijk}^2 = 0 \tag{2.8}$$

Note that since $T_{A_1}, T_{B_1}, T_{C_1}$ and F_{ijk} are known, this represents a second-order polynomial in one variable, T_{ijk}. For its roots to be real, its discriminant $D \equiv b^2 - 4ac$ needs to be non-negative, where a, b, c represent the coefficients of the various orders of T_{ijk} in the standardised representation of a quadratic equation, i.e. $aT_{ijk}^2 + bT_{ijk} + c = 0$. For $D \geq 0$, (2.8) can be solved for its largest root, the only valid solution if monotonicity is to hold, using the standard solution formula for quadratic equations. If this root meets the condition $T_{ijk} \geq T_{C_1}$, a new valid estimate of T_{ijk} has been found; otherwise, or if $D < 0$, one of the 6-connectivity neighbours of vertex $(i\Delta x, j\Delta y, k\Delta z)$ features too large an arrival time to be considered in the solution and (2.8) is to be simplified to

$$(T_{ijk} - T_{B_1})^2 + (T_{ijk} - T_{A_1})^2 - F_{ijk}^2 = 0 \tag{2.9}$$

Again, if the discriminant of this quadratic equation is negative or its largest root violates the condition $T_{ijk} \geq T_{B_1} (\geq T_{A_1})$, T_{B_1} is to be dropped from consideration; otherwise the new T_{ijk}-estimate has been found. In the former case, it follows

$$T_{ijk} = T_{A_1} + F_{ijk} \tag{2.10}$$

as the new arrival time estimate at vertex $(i\Delta x, j\Delta y, k\Delta z)$.

The above represents the sequential solution process for the case of ALIVE neighbours in each grid direction. In case ALIVE neighbours are found in only two of the three directions, the updating process starts with the consideration of equation (2.9) instead. This also represents the starting point of the updating process when performing Cartesian Fast Marching on a 2D grid. Finally, if only T_{A_1} is known, the updating process narrows down to the immediate solution of (2.10). With this updating process implemented, all key elements of Algorithm 1 are in place.

These elements can be re-used to a degree in the implementation of Fast Marching for surfaces in point cloud form discussed next.

2.5.2 Fast Marching for surfaces in point cloud form

Apart from the need to compute Ω_P^r and to subsequently restrict processing to those grid vertices located inside Ω_P^r, the implementation requirements of Cartesian Fast Marching for point clouds and conventional Fast Marching are identical.

The computation of Ω_P^r is implemented by using conventional Fast Marching in the embedding space. That is, Ω_P^r is constructed by propagating fronts simultaneously or sequentially at unit speed from each input point outwards until their extent corresponds to the given constant radius r; r may, for example, be estimated from the sampling resolution of the device used to acquire the surface data. Grid vertices inside of Ω_P^r are assigned the state VALID, with subsequent processing being restricted to these valid vertices only. The design of the Cartesian grid as a lookup table proves beneficial in this context. By removing all invalid grid vertices from the lookup table, memory demand is reduced to a fraction of the memory requirements of the object's discretised bounding volume. See, for example, Figure 2.8. Thus, the relatively more memory-demanding offset band computation for a particular object surface can be performed as a separate pre-processing step, with the resulting lookup table of valid grid vertices made available alongside the surface point cloud for subsequent processing.

To compute an adaptive offset band $\Omega_P^{r_i}$, the variable offset ball radii, r_i, need to be determined first as part of the one-off pre-processing step. The required adjacency information is obtained by computing the enhanced k nearest neighbourhood eNN_{p_i} of the point p_i under consideration (Section 4.5.2). Once this local proximity information is available, the algorithm determines the Euclidean distance between p_i and its neighbour $q \in eNN_{p_i}$ farthest away from p_i. If $r_i = d(p_i, q)$ is larger than any radius currently associated with p_i and q, r_i is set as the new radii of the offset balls centred at p_i and q; otherwise, r_i is ignored. Once all variable radii r_i have been determined this way, the corresponding (valid) grid vertices are included in the band by propagating fronts with extent r_i from each p_i outwards. This technique is discussed in more detail in Section 4.3.2 where it is used in the context of the resampling and hole-filling of non-uniformly distributed point clouds.

With the offset band computation in place, the rest of the implementation of Fast Marching for point clouds follows the presentation in Section 2.5.1. Since the offset band pre-

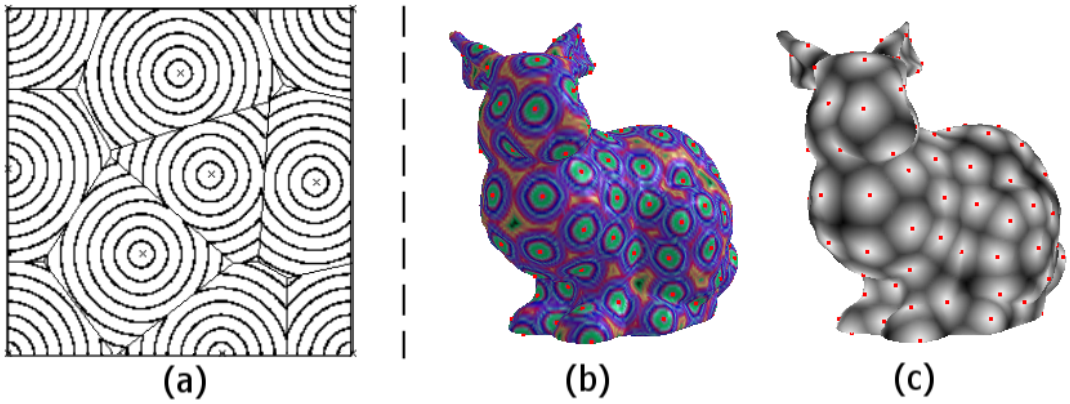

Figure 2.16: Wave propagation for the computation of discrete geodesic Voronoi diagrams. By propagating waves for geodesic distance mapping purposes from the crossed (a) and red (b) generator points outwards respectively, intrinsic Voronoi partitionings of a planar domain (a) and a triangulated surface (c) are obtained.

processing step results in a lookup table consisting of grid vertices inside Ω_P^r only, conventional Cartesian Fast Marching across this grid is automatically restricted to vertices in Ω_P^r. That is, explicit testing for the membership of a grid vertex in Ω_P^r is not required (compare with Algorithm 3).

Amongst other things, I employ the Fast Marching implementations presented above for the computation of geodesic Voronoi diagrams discussed next.

2.5.3 Geodesic Voronoi diagrams

The intrinsic Voronoi partitioning of a surface is implemented by approximating the continuous geodesic Voronoi diagram of point set P with geodesic distance maps, i.e. $VD(P)$ is generated following an expanding waves view. That is, by analogy to the dropping of pebbles in still water, circular fronts move across the surface from the points of impact. The locations where wave fronts meet define the geodesic Voronoi diagram of the points of impact. Figure 2.16 gives both a planar and a triangular mesh-based example for geodesic Voronoi diagrams computed with the help of wave propagation across the surface.[6]

The wave propagation is discretised and simulated accurately by solving (2.1) in the Cartesian grid, (2.4) in the unstructured grid and (2.7) in the point set surface case respectively. These wave propagations in the form of partial intrinsic distance maps are computed for all points on the surface simultaneously using the Fast Marching techniques for intrinsic distance mapping discussed in Section 2.3.

[6]The triangular mesh-based example was visualised with the help of public domain software [127].

Alternatively, a geodesic Voronoi diagram may be computed *incrementally* by partial intrinsic distance mapping from each point $p_i \in P$ outwards thereby obtaining the Voronoi regions $R(p_i, P)$ sequentially rather than simultaneously.

Irrespective of whether a discrete geodesic Voronoi diagram is computed using simultaneous or incremental partial intrinsic distance mapping, the implementation of the front propagation for Cartesian grids, triangulated or point set surfaces is readily available in the form of my implementations of conventional Cartesian Fast Marching (Section 2.5.1) and Fast Marching for surfaces in point cloud form (Section 2.5.2). My implementation of Fast Marching for triangulated surfaces is discussed in Section 3.7.1. These implementations are used to propagate a front from a point $p_i \in P$ outwards until loci of equal arrival times originating from points $p_j \in P$, $p_i \neq p_j$, are encountered. These loci represent the Voronoi edges or vertices of Voronoi region $R(p_i, P)$ shared with its neighbouring Voronoi regions in $VD(P)$. To support the applications discussed in the subsequent chapters, only the vertices of $VD(P)$ are explicitly required and thus stored for further processing.

This completes the discussion of my implementations of those algorithms used repeatedly in this thesis.

2.6 Summary and discussion

In this chapter, I presented various concepts, definitions and implementations used throughout this thesis, in particular recent advances in Fast Marching level set methods for weighted intrinsic distance mapping, geodesic Voronoi diagrams and neighbourhood concepts for point-sampled geometry.

The main contributions of this chapter are my

- intrinsic natural neighbourhood concept for point-sampled geometry.
- implementation of Fast Marching on Cartesian grids.
- implementation of Fast Marching for surfaces in point cloud form.
- implementation of discrete geodesic Voronoi diagram computation.

Fast Marching on Cartesian grids has been implemented and widely-used for a number of years [145]. By contrast, Fast Marching for surfaces in point cloud form has only been put forward and implemented for the first time relatively recently [104]. With the help of

Fast Marching for surfaces in point cloud form, discrete geodesic Voronoi diagrams can now be computed efficiently across point-sampled geometry.

The intrinsic natural neighbourhood concept inherently defines proximity in terms of geodesic interpoint distances. It neither requires the computation of local ball radii as in the case of Euclidean ball neighbourhoods nor the solution of least squares problems and the associated complications as may be encountered in the case of k nearest neighbourhoods, enhanced or otherwise [64]. It is experimentally found to avoid the pitfalls of point-based neighbour determination more reliably than existing extrinsic neighbourhood concepts. For intrinsic natural neighbourhoods to be well-defined, the underlying geodesic Voronoi partitioning of the surface needs to be well-defined. The Fast Marching techniques discussed in this chapter allow to compute weighted intrinsic distances on surfaces in triangular, implicit or point cloud form. This permits the computation of equal distance curves and thus bisectors between points on the surface. As a result, we have a meaningful way of computing geodesic Voronoi diagrams on surfaces in the most widely-used forms of representation. From these intrinsic surface partitionings, intrinsic natural neighbourhood information is readily available.

For not particularly dense point sets, intrinsic natural neighbourhoods can be relatively small. This can be problematic for the support of the computation of differential properties such as surface normals: A point's 2- or 3-ring intrinsic natural neighbours may have to be collected to obtain sufficiently many neighbours for the normal estimation to be robust. Unlike their 1-ring counterparts, these 2- or 3-ring neighbours are not immediately available and would require the traversal of the diagram. In the context of this thesis, intrinsic natural neighbourhoods are not used for this purpose.

Although the intrinsic natural neighbourhood concept has proven experimentally to be useful for the purposes of this thesis, a theoretical study of its properties and those of existing extrinsic neighbourhood concepts would be useful. Such a study, should, for example, provide results on how well a point neighbourhood can capture the local topology of the underlying surface even if the point set is not of global ϵ-density [9, 90] (Appendix B). The required advances in computational geometry for point-sampled geometry are beyond the scope of this thesis.

Chapter 3

Intrinsic point sampling of surfaces

Point sampling generally deals with the approximation of continuous functions by a limited number of discrete sample points. The problem of placing point samples uniformly or subject to, for example, feature-sensitivity or approximation error measures is ubiquitous in both image and geometry processing. Typical image sampling tasks include image synthesis/rendering [55, 56] and compression [155]. Geometry-related sampling problems include mesh [58] and point cloud simplification (Chapter 4), remeshing [6, 163], implicit surface polygonisation [162] and geometry rendering [139, 167, 179]. Note that geometry sampling may involve surfaces in three or higher dimensions, typically represented in the form of triangle meshes, implicit functions or point clouds.

In this chapter, I present a point sampling algorithm with sampling density guarantee that operates truly intrinsically using weighted intrinsic distance mapping. A sample density guarantee is desirable given the fact that the distribution of the samples determines their anti-aliasing and noise properties [43]. It is also needed in the context of applications such as point-based rendering to ensure the sufficient coverage of the surface [59]. The algorithm's intrinsic nature implies that it operates inherently sensitively to the geometry. The algorithm is formulated as a template, or meta algorithm, which is generic enough to be equally well instantiable for both images and high-dimensional geometry in the most widely-used forms of representation for both uniform and adaptive sampling tasks. This is shown in this and subsequent chapters using the Fast Marching framework.

The following section reviews relevant existing point sampling techniques. The farthest point sampling principle is reviewed in detail in Section 3.2 and then generalised to my algorithm template in Section 3.3. The modular structure of this template is discussed in Section 3.4. Section 3.5 gives the algorithm's sampling density guarantee. Since most sampling concepts have originated in an image sampling context, I present and analyse

my algorithm instantiation for planar domains in Section 3.6. The section analyses the algorithm's computational complexity, the generated sampling pattern and its usefulness with respect to the original farthest point sampling algorithm [47]. To demonstrate the algorithm template's generic nature, Section 3.7 presents its instantiation for triangulated surfaces. Since this thesis is predominantly concerned with the intrinsic processing of surfaces in point cloud form, this discussion is relatively brief. Point cloud farthest point sampling is discussed in the context of intrinsic meshless surface simplification in Chapter 4.

This chapter is based on Moenning and Dodgson [110, 111].

3.1 Related Work

I start with the review of point sampling principles which have proved useful for the uniform and adaptive point sampling of both images and geometry in various forms of representation.

Grid-based sampling techniques are based on the idea of partitioning the sampling domain by a grid data structure. The domain is then sampled directly at or relative to the grid vertices. Grid-based sampling methods have been devised for image processing purposes in the form of regular grid, stratified, jittered and N-rooks, or Latin hypercube, sampling. See Dippe [43], Cook [34], Glassner [56] or Shirley [150] for details. In the context of geometry processing, grid-based point sampling methods include clustering or voxelisation methods for triangular meshes and point clouds and uniform or adaptive space partitioning techniques supporting implicit surface polygonisation. See, e.g. Lindstrom [96], Low and Tan [100] and Rossignac and Borrel [135] (meshes), Kalaiah and Varshney [78] and Nehab and Shilane [116] (point clouds) and Hall and Warren [67], Lorensen and Cline [99] and Velho et al. [162] (implicits). Although grid-based point sampling techniques therefore exist for images and different geometry representations, their details vary substantially and no single, generally applicable technique exists. Overall, grid-based techniques tend to be relatively simple at the expense of a lack of feature-sensitivity for the control of sampling distributions and relatively high memory demands caused by the discretisation of the domain's bounding box.

Particle simulation-based sampling techniques are of a more generic nature. Particle simulation typically starts with a set of particles distributed across the object at random. This set is exposed to repulsion and/or attraction forces which may vary with local curvature estimates and other object attributes [73]. Relaxation of the system leads to the

drifting of the particles into their final position of lowest energy on or near the object thereby yielding a uniform or feature-sensitive resampling. Similar particle-based point sampling methods have been devised for surfaces in mesh, implicit and point cloud form. See, e.g. Turk [159] (meshes), Figueiredo et al. [48] and Witkin and Heckbert [171] (implicits) and Pauly et al. [124] (point clouds). Particle simulation allows for good control of the sampling distribution at the expense of the computationally demanding processing of energy equations and a potentially slowly converging process.

Eldar et al. [47] introduce the notion of *farthest point sampling* in an image sampling context. Farthest point sampling is based on the idea of repeatedly placing the next sample point in the middle of the least-known area of the sampling domain. That is, the next sample point is placed farthest away from all existing sample sites. Ulichney [160] suggests a similar idea for digital halftoning purposes. Shahidi et al. [148] slightly modify the farthest point sampling idea of Eldar et al. [47] to improve the performance of their sampling scheme also in a halftoning context. Since in the case of halftoning only distances between points on an integer grid need to be considered, their idea is to use a lookup table as opposed to a Voronoi diagram for the quick detection of the next farthest point sample. Grundland et al. [60] develop a farthest point sampling-based algorithm for the automated stylised rendering of a multiresolution image representation. They aim at giving the user creative control over the stylisation/rendition of a compressed image. The main application of their algorithm is in the context of progressive image compression. As regards triangulated surfaces, farthest point sampling has been widely used for remeshing purposes. See, e.g. Chew [29], Peyré and Cohen [127] and Ruppert [138]. Peyré and Cohen [128] use the same technique for the segmentation of triangular meshes. Boissonnat and Oudot [21] generalise the farthest point sampling concept to a degree by solving the sample localisation problem *extrinsically* using incremental restricted Euclidean Delaunay triangulation [20] (Appendix B) in the embedding space. As a result, they obtain a sampling method with guarantees for smooth surfaces in both triangular mesh and implicit form. The theoretical guarantees follow from the fact that their algorithm can generate ε-samples in the sense of Amenta and Bern [8] (Appendix B). Although this sampling condition is of great theoretical value, its practical usefulness is limited. Approximations of a surface point's distance to the medial axis may be computed in practice with the help of the "poles" [8] (Appendix B) of the current sample set's Euclidean Voronoi diagram at the loss of Boissonnat and Oudot [21] algorithm's theoretical guarantees and execution and memory efficiency. Also note that Boissonnat and Oudot's [21] extension of the farthest point sampling method does not apply to the increasingly important case of surfaces in point cloud form.

I generalise the farthest point sampling concept more generically by solving the sample point localisation problem directly and intrinsically on the object under consideration rather than using any global combinatorial data structures in its embedding space. The approach is presented following the detailed review of the original farthest point sampling algorithm introduced by Eldar et al. [47].

3.2 Farthest point sampling

In the following, I summarise the reasoning underlying the farthest point sampling approach introduced by Eldar et al. [47] in an image processing context for both the uniform and non-uniform, adaptive sampling case.

Starting with the uniform sampling case, Eldar et al. [47] consider an image as represented by a continuous stochastic process featuring constant first and second order central moments with the covariance decreasing exponentially with spatial distance. That is, given a pair of sample points $s_i = (x_i, y_i), s_j = (x_j, y_j)$, the correlation of the points' image intensities, $I(s_i)$, $I(s_j)$, is assumed to decrease exponentially with the Euclidean distance between the points,

$$E(I(s_i), I(s_j)) = \sigma^2 e^{-\lambda d(s_i, s_j)}$$

Based on their linear estimator for the image interpolation, the authors subsequently put forward the following representation for the expected mean square (reconstruction) error, i.e. the deviation from the ideal image resulting from estimation error, after the Nth sample

$$\varepsilon^2(s_0, \ldots, s_{N-1}) = \iint \sigma^2 - Z^T W^{-1} Z \, dx \, dy$$

with covariance matrix

$$W_{ij} = \sigma^2 e^{-\lambda d(s_i, s_j)}$$

and variance matrix

$$Z_i = \sigma^2 e^{-\lambda d(s_i, s)},$$

for $0 \leq i, j \leq N$ and with $s = (x, y)$. The assumption of stationary first and second order central moments has therefore yielded the result that the expected mean square error depends on the location of the Nth sample only rather than both its location and its image intensity value. Since stationarity implies that the image's statistical properties are

spatially invariant and given that intensity correlations decrease with distance, uniformly choosing the Nth sample point to be that point which is farthest away from the current set of sample points therefore represents the optimal sampling approach for the minimisation of the expected reconstruction error [47]. That is, the next sample, s_N, is to be placed subject to the following criterion[1]

$$d(s_N, S) = \max_{q \in \mathcal{D}} (\min_{0 \leq i \leq N-1} d(q, s_i)), \tag{3.1}$$

with $s_i \in S$; S is the set of existing samples; $\mathcal{D}$ represents the domain of the image.

This sampling approach is intimately linked with the incremental construction of a (bounded) Voronoi diagram over the image domain. To see this, note that the point farthest away from S is represented by the centre of the largest circle empty of any site $s_i \in S$ [30]. Eldar et al. [47] show that in the case of farthest point sequences, the centre of such a circle is repeatedly given by a vertex of the bounded Voronoi diagram of S, bounded $VD(S)$. A similar result can also been found in the context of facility location and group coordination problems. In these cases, (3.1) is interpreted as a disk covering problem, namely how to cover a region with possibly overlapping disks of equal minimum radius. This problem is shown to be solved by choosing the farthest vertex in the bounded Voronoi diagram of existing sites as disk centre [35].

Thus, incremental bounded Voronoi diagram construction provides sample points *progressively*, i.e. a representation of the entire domain is available from the start of the sampling process which is subsequently uniformly refined in detail. The algorithm is found to support a high sample acquisition rate and to produce sample sets featuring a blue noise power spectrum and thus excellent anti-aliasing properties [34, 107, 160].

From visual inspection of images it is clear that usually not only the sample covariances but also the sample means and variances vary spatially across an image. When allowing for this more general variability and thus turning to the design of a non-uniform, adaptive sampling strategy, the assumption of image intensity covariances decreasing, exponentially or otherwise, with point distance remains valid. However, since Voronoi diagrams in non-uniform metrics may lose favourable properties such as Voronoi region connectedness [120], Eldar et al. [47] consider the detection of the vertices of such a diagram impractical and opt for augmenting their model by application-dependent weighting schemes for the vertices in the Euclidean Voronoi diagram. More specifically, they compute the weight for a vertex of the Euclidean Voronoi diagram with the help of local bandwidth estimation [23] across the

[1] This min-max criterion was considered earlier by Gonzalez [57] and also Mitchell [107]. However, Eldar [46] was first to thoroughly analyse the idea and to exploit its close relationship with incremental bounded Voronoi diagram computation in an image processing context.

neighbourhood of the three closest sample points of each vertex. This way the character of the image in the vicinity of the existing samples is evaluated. The estimates are used in the design of various weight functions of the Voronoi vertices presented in Eldar [46].

This approach is non-optimal due to, firstly, the need to incorporate a factor in the weighting schemes reflecting a vertex' minimum Euclidean distance from the current sample set. Following experimentation, Eldar et al. [47] decide to include such a factor multiplicatively. The impact of this choice on the nature of the sampling distribution is not analysed. Secondly, in general, arbitrary distance metrics may be used in the computation of Voronoi diagrams provided the resulting dominance regions are well-behaved, i.e. overlapping on their bisectors only and representing a tessellation of the domain [120]. The modelling of non-uniform distance metrics with the help of weighted intrinsic distance mapping to control the shape and size of intrinsic Voronoi regions represents one way of approaching this problem with the edges/vertices of this diagram remaining tractable [83, 85]. In the following section, I utilise this idea for the design of a generalised farthest point sampling algorithm which may be used for *both* uniform and adaptive sampling of *both* planar domains and manifold surfaces.

3.3 The generic intrinsic point sampling algorithm

I re-formulate Eldar et al. [47] farthest point sampling idea using weighted intrinsic distance mapping. As discussed in detail in Chapter 2, such distance maps may be computed by propagating equal distance contours across the surface. When simultaneously propagating such contours with unit velocity from a set of points on a surface outwards, the geodesic Voronoi diagram of these points is obtained. By allowing for different weights/speeds at each point during the propagation, a geodesic Voronoi diagram in a non-uniform metric is computed. Partial weighted intrinsic distance mapping also allows to generate such diagrams incrementally. The combination of these aspects represents the basic idea underpinning the intrinsic farthest point algorithm template presented in Algorithm 4.

Given an initial, random sample set S, $n = \|S\| \geq 3$ and weight function $F > 0$, bounded $VD(S)$ is computed using *simultaneous* weighted intrinsic distance mapping. The vertices of the bounded $VD(S)$ represent the initial set of farthest point candidates which are stored in a priority queue sorted in descending order of arrival time. The iterative sampling then amounts to the extraction of the root from the queue yielding the location of s_{n+1}. The subsequent computation of bounded $R(s_{n+1}, S')$ by *partial* weighted intrinsic distance

Input: Weight function $F > 0$. Sample budget N or refinement threshold $\rho > 0$ (termination condition).
Output: Sample set S' following from the enforcement of the termination condition.

---

0 *** *Initialisation* ***
1 Randomly select an initial set $S \geq 3 (= n)$ of sample points;
2 Compute bounded $VD(S)$ using simultaneous weighted intrinsic distance mapping;
3 Insert the vertices of $VD(S)$ in a priority queue;
4
5 *** *Sampling* ***
6 **REPEAT**
7 Extract the root from the queue to obtain s_{n+1};
8 $S' = S \cup \{s_{n+1}\}$;
9 Compute bounded $R(s_{n+1}, S')$ (and thus bounded $VD(S')$) using partial weighted intrinsic distance mapping;
10 Remove any obsolete vertices of the neighbouring regions of bounded $R(s_{n+1}, S')$ from the queue;
11 Insert the Voronoi vertices of bounded $R(s_{n+1}, S')$ in the queue;
12 **UNTIL** $\|S'\| = N$ or ρ has been met;

Alg. 4: Intrinsic farthest point sampling algorithm template in pseudocode.

mapping gives the bounded $VD(S')$. The vertices of this region are added to the queue of farthest point candidates. Those Voronoi vertices of the neighbours of bounded $R(s_{n+1}, S')$ which have become obsolete due to the expansion of bounded $R(s_{n+1}, S')$ into their regions are removed from the queue. This sampling process, illustrated in Figure 3.1, is repeated until the user-controlled sampling budget is exhausted or a refinement condition has been met.

The refinement condition is formulated in the form of a density condition ρ which I show in the following section to imply deterministic bounds on sample distances. More specifically, the user controls the maximum distance from the next farthest point candidate to the current sample set S, i.e. the radius $\rho > 0$ of the largest empty geodesic circle on the object. The sample set is refined until the next farthest point candidate's arrival time is no longer larger than or equal to this radius indicating that S' has become sufficiently dense. That is, the user controls the radius of the disks covering the sampling domain. As an alternative to the selection of a global value for ρ, the density condition can be formulated as a function of local object properties provided $\rho > 0$ holds true throughout. The effect of different values for ρ is illustrated in Figure 3.2.

From $\rho > 0$, it is clear that the algorithm terminates: As a result of the density condition's enforcement, sample points cannot be placed arbitrarily closely to each other and thus cannot be infinitely many.

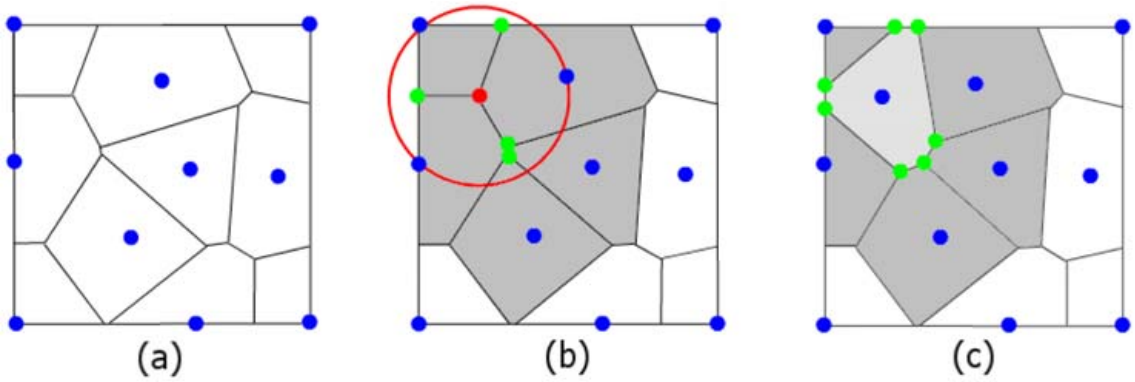

Figure 3.1: Algorithm 4 uses the fact that the centre of the (red) largest empty circle coincides with a vertex of bounded $VD(S)$. That is, sample $s_{n+1} \in S'$ farthest away from the existing set of (blue) sample points coincides with a (red) vertex of bounded $VD(S)$ ((a) and (b)). During the computation of the (light grey) bounded $R(s_{n+1}, S')$, (green) vertices of (dark grey) neighbouring Voronoi regions of $R(s_{n+1}, S')$ which now belong to $R(s_{n+1}, S')$ are removed from the queue of farthest point candidates. The (green) Voronoi vertices of bounded $R(s_{n+1}, S')$ are added to the queue ((b) and (c)). This process continues until the user-controlled termination condition is met.

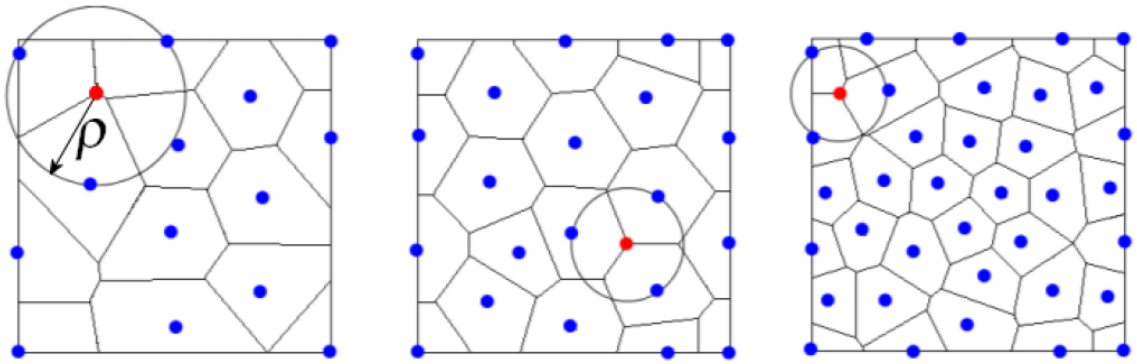

Figure 3.2: Effect of different (global) values for ρ. The user controls the maximum distance from the next (red) farthest point candidate to the (blue) sample set, i.e. the radius ρ of the largest empty circle in the sampling domain. This in turn bounds interpoint distances.

The algorithm allows for the computation of bounded geodesic Voronoi diagrams in non-uniform metrics by simply varying the weight function F with arbitrary local properties. This way distances between points in highly variable regions will be larger than those of points equally-spaced in the Euclidean sense but located in areas of low variance. Thus, the farthest point sampling principle is extended to non-uniform, adaptive sampling in the more natural and general manner hinted at by Eldar et al. [47] as the preferable approach. F may either be computed on-the-fly during intrinsic distance mapping or pre-computed and passed on to the algorithm in the form of an importance map.

3.4 Modular algorithm structure

As illustrated in Figure 3.3, the algorithm is designed as a collection of self-contained modules. This way, a library of module models can be built up, each representing an implementation for a different surface representation. This toolbox can then be stored in the form of a library of code modules.

The Fast Marching module implements weighted intrinsic distance mapping across a structured or unstructured grid represented by the grid module implementation. The Fast Marching module is also linked with the weight function module which operates as an oracle providing non-uniform propagation speeds, possibly upon provision of information stored at the relevant grid element. The heap module provides the priority queue implementation and is linked with the grid module which manages a grid element's heap indices and the Fast Marching module which processes its elements. The heap elements hold their positions in the grid.

By providing implementations of the Fast Marching, grid and weight function modules as required, a toolbox for the point sampling of surfaces in the most widely-used forms of representation is obtained. This toolbox may be used for the flexible uniform or adaptive intrinsic point sampling of surfaces with the sampling density guarantee discussed next.

3.5 Sampling density guarantee

A simple guarantee in the spirit of Eldar et al. [47] is given in terms of the user-controlled refinement condition ρ on the intersample distances of samples placed by Algorithm 4.

Definition 1 As discussed in Section 3.2, the centre of the largest geodesic circle empty of any sample points coincides at any one stage of the sampling process with a vertex v of (bounded) $VD(S')$. For ease of exposition, set r_{max} equal to the radius of this circle *at the end* of the sampling process, i.e. $r_{max} = \max_{s \in S} d_M(v, s) = \rho$. Denote as r^j_{max}, the radius of the largest geodesic circle empty of the *first j sample points*. S^j represents the set of j samples. Without loss of generality, it is assumed that 1(a) below is not violated by the initial set of three sample points. This may be guaranteed by choosing the first sample point at random and then repeatedly placing a sample at the point farthest away in the existing samples' distance maps.

Theorem 1

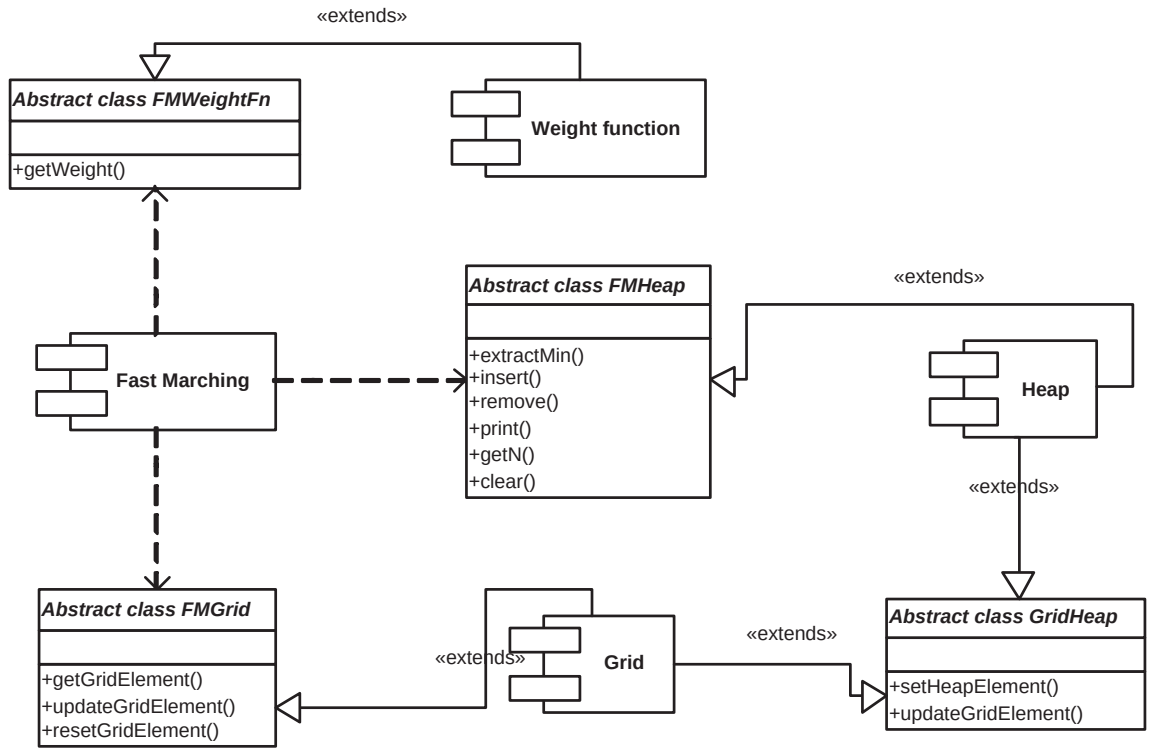

Figure 3.3: Modular design of Algorithm 4 in the form of a UML component diagram [52].

(a) For the intrinsic distance between samples s_i, $s_j \in S'$, $i \neq j$, $d_M(s_i, s_j) \geq \rho$.
(b) For any pair of neighbouring samples $s_i, s_j \in S'$, $d_M(s_i, s_j) \leq 2\rho$.

Proof
(a) Note that with increasing sample size, r^j_{max} will not increase,

$$r_{max} \leq r^j_{max}, \tag{3.2}$$

for $j < N(= \|S'\|)$. Let s_i, s_j denote two samples on the geodesic circle of v with j sampled after i. By the definition of the algorithm, s_j is placed at a vertex with distance $r^{j-1}_{max} = \max_{s \in S^{j-1}} d_M(v, s)$ from S^{j-1}. Thus, for any s_i with $i < j$, $d_M(s_i, s_j) \geq r^{j-1}_{max}$. From (3.2), $r_{max} \leq r^{j-1}_{max}$ so that $d_M(s_i, s_j) \geq r_{max} = \rho$. □

(b) Consider a Voronoi vertex v shared by two neighbouring sample points s_i, s_j. By definition, v is equally far away from both s_i and s_j. The geodesic distance is a metric on M. Thus, it satisfies the metric triangle inequality and $d_M(s_i, s_j) \leq 2d_M(v, s_i)$. Since $d_M(v, s_i) \leq r_{max} = \rho$, $d_M(s_i, s_j) \leq 2\rho$. □

It follows that the samples cover the domain irregularly uniformly without clustering

or leaving large holes. Apart from being desirable from an anti-aliasing point of view, this guarantee ensures the support of meaningful further processing [34, 43].

The properties of the sampling distribution generated by this algorithm are analysed in more detail in the following section. More specifically, I comparatively evaluate the properties of the algorithm instantiation for planar domains with respect to those of the original farthest point sampling algorithm by Eldar et al. [47].

Any instantiation of Algorithm 4 primarily needs to address the question of how to efficiently perform partial weighted intrinsic distance mapping for incremental (bounded) geodesic Voronoi diagram computation. As described in the following, I utilise the computationally optimal Fast Marching methods of Section 2.3 for this purpose. Henceforth, I refer to the resulting algorithm instantiations as Fast Marching farthest point sampling (FastFPS) algorithms.

3.6 FastFPS of planar domains

In the following, Section 3.6.1 discusses my implementation of Algorithm 4 for planar domains. Section 3.6.2 analyses the implementation's computational complexity. The generated sampling pattern is analysed with an eye to the properties of the original farthest point sampling algorithm [47] in Section 3.6.3. Section 3.6.4 concludes this chapter with experimental results for uniform and adaptive FastFPS image sampling.

3.6.1 Implementation of the generic algorithm for planar domains

Any instantiation of the algorithm template needs to provide implementations for the modules listed in Section 3.4. In the case of planar domains, these modules consist of Fast Marching for 2D Cartesian grids, my implementation of which is discussed in Section 2.5.1. The grid module is implemented in the form of a 2D vector. The priority queue implementation is discussed alongside the other remaining algorithm details in the following.

The FastFPS implementation proceeds with the selection of the first sample point, s_1, at random; s_2 and s_3 are chosen as the vertices farthest away in $T_M(s_1)$ and $T_M(s_2)$ respectively. $T_M(s_1)$ and $T_M(s_2)$ follow from partial weighted intrinsic distance mapping from s_1 and s_2 outwards. Initialisation is completed with the computation of the discretised

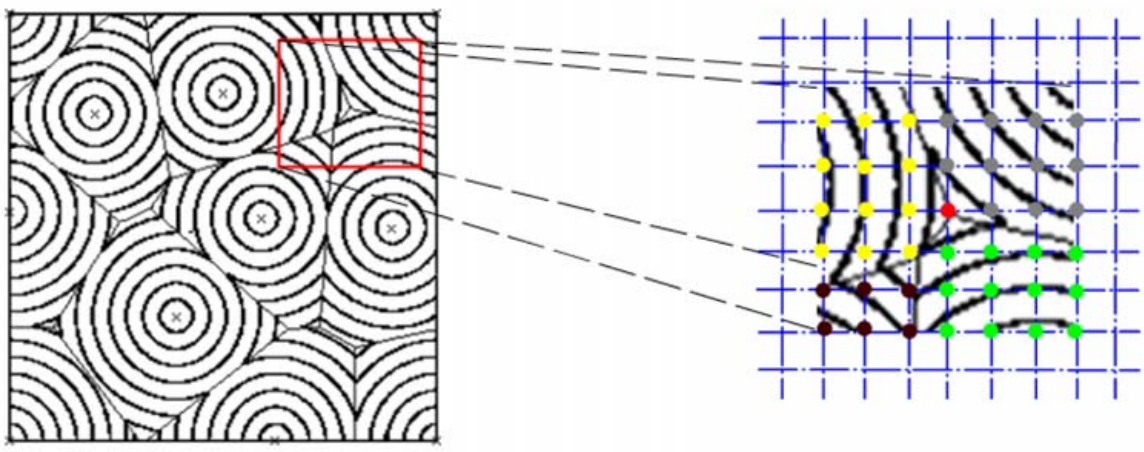

Figure 3.4: Detection of intrinsic Voronoi vertices, and thus farthest point candidates, during front propagation. Take the red grid vertex to be the (in this case, last) member of a front to be assigned its final arrival time. It is surrounded by ALIVE grid vertices belonging to three different Voronoi regions as indicated by their labels of origin, here represented by different colours, grey, yellow, green and brown respectively. Since each grid vertex maintains this information (Figure 2.14), the red vertex can be identified as a Voronoi vertex as part of its arrival time computation which necessarily accesses its ALIVE grid neighbours. Any additional grid traversal is not required.

bounded $VD(S)$ by simultaneous propagation of circular fronts originating from s_1, s_2 and s_3. The vertices of $VD(S)$ are detected (Figure 3.4), stored in a priority queue and labelled as Voronoi vertices in the grid during front propagation using the Fast Marching module of Section 2.5.1. As illustrated in Figure 3.5, Voronoi vertices on the boundary are characterised by two entering waves, with internal Voronoi vertices being characterised by three entering waves.

The priority queue is implemented in the form of a max-heap, i.e. an instantiation of the templated heap class of Section 2.5.1 (Figure 2.14) with its elements keyed by their *negative* arrival times. As a result, the Voronoi vertex farthest away is located at the root. The re-heapification method's friendly access to the grid data structure allows for the updating of the max-heap index stored at the relevant grid vertex following root extraction (line 7), vertex removal (line 10), or vertex insertion (line 11). Both this max-heap and the min-heap used in the context of Fast Marching make up the heap module of this instantiation.

Partial weighted intrinsic distance mapping for the computation of bounded Voronoi region $R(s_{n+1}, S')$ and thus $VD(S')$ (line 9) is implemented by propagating a front locally from s_{n+1} outwards until loci of equal arrival time values, i.e. grid vertices on the Voronoi edges of bounded $R(s_{n+1}, S')$, are encountered (Figure 3.5). Existing Voronoi vertices passed during the propagation are removed from the max-heap using their max-heap in-

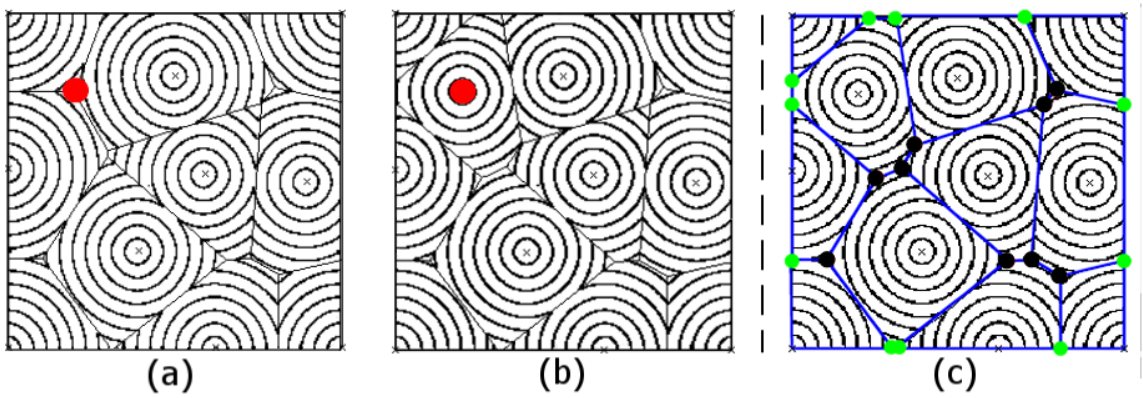

Figure 3.5: Partial intrinsic distance mapping from the new (red) sample point s_{n+1} outwards (a) yields $R(s_{n+1}, S')$ (b) (cp. with Figure 3.1). The (green) vertices of bounded $VD(S')$ located on the boundary are characterised by two entering propagation waves (c). Any (black) internal Voronoi vertices can be identified by the fact that they are entered by three different waves. Existing sample points are marked by crosses.

dices stored at the corresponding grid vertices (line 10). The Voronoi vertices of bounded $R(s_{n+1}, S')$ are detected, stored in the max-heap and labelled as during initialisation (line 11).

The enforcement of refinement condition ρ (line 12) simply amounts to controlling for the arrival time of the extracted max-heap root being greater than or equal to ρ.

This completes the implementation of Algorithm 4 for the planar domain case.

3.6.2 Computational complexity

As regards my implementation's efficiency, extracting the root from (line 7), inserting into (lines 3, 11), updating of (line 9) and removing from (lines 2, 9, 10) the max- and/or min-heap with subsequent re-heapifying are $O(\log W_1)$ and $O(\log W_2)$ operations respectively, where W_1 represents the number of elements in the max-heap and W_2 denotes the number of elements in the min-heap. W_1 and W_2 are $O(N)$, N representing the number of grid vertices of the 2D Cartesian grid. The accessing of existing max- and min-heap entries is $O(1)$ due to the storage of the heap indices in the grid. The detection of the Voronoi vertices is a by-product of the $O(N \log N)$ front propagation (lines 2, 9). This $O(N \log N)$ process is performed up to N_2 times, where N_2 equals $\|S'\|$ after algorithm termination (line 12). N_2 is $O(N)$ yielding a running time of $O(N^2 \log N)$.

It is important to note in this context that the size of W_2, i.e. the size of the narrow band during partial intrinsic distance mapping, decreases with increasing sample size moving

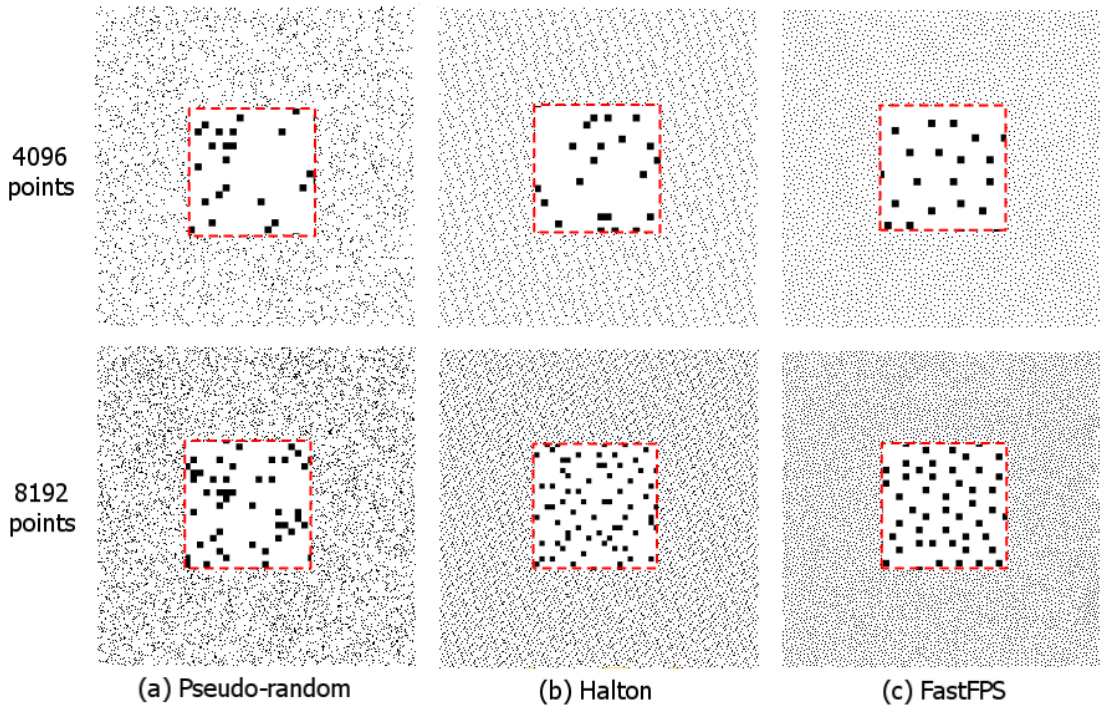

Figure 3.6: Sample sets produced by pseudo-random sampling are characterised by clustering and holes undermining meaningful further processing (a). Incrementally computable quasi-random sequences such as Halton avoid the latter problem but still produce undesirable sample distributions (here: samples may be placed arbitrarily closely to each other) (b). Uniform FastFPS places samples within deterministic minimum and maximum distances of each other thereby preventing both undersampling and clustering (c). The sparse squares shown in the middle of each sample set represent blown-up views of the generated sample distributions.

min-heap re-heapifying closer to $O(1)$ and making the $O(N \log N)$ front propagation close to linear in complexity. Also, typically $N_2 \ll N$. Thus, the run-time behaviour tends to be considerably more favourable in practice. This observation is confirmed by the experimental results given in Section 3.6.4 after the following analysis of uniform farthest point sample distributions.

3.6.3 Analysis of the generated sampling distribution

A good uniform sample distribution is expected both to provide similar amounts of information about different parts of the image whilst avoiding holes or clustering and to be irregular so as to minimise aliasing effects.

As regards the former aspect, the nature of the sample distribution generated by uniform FastFPS of planar domains is illustrated in Figure 3.6. The figure presents a uniform FastFPS point set alongside realisations of two independently uniformly dis-

tributed (pseudo-)random variables,[2] and a quasi-random Halton sequence [68]. A Halton sequence defines the ith point in an arbitrary-length, m-dimensional sequence as $u_i = (\phi_{b_1}(i), \ldots, \phi_{b_m}(i))$, where the bases $b_1, \ldots, b_m$ are typically chosen as the first m prime numbers to avoid component correlations and the Van Corput sequence $\phi_b(i)$ is defined as $\phi_b(i) = \sum_{j=0}^{\infty} a_j b^{-j-1}$, with a_j being the coefficients in the b-ary expansion of i, $i = \sum_{j=0}^{\infty} a_j b^j$. The resulting points lie inside the m-dimensional unit cube $[0,1)^m$. Halton sequences are an example of low-discrepancy deterministic point sequences, i.e. sequences whose discrepancy is significantly smaller than the discrepancy of a typical set of randomly distributed points. Discrepancy is here understood as a measure of the deviation of the sample set from the uniform distribution in the unit cube [80, 150]. Thus, by choosing a two-dimensional Halton point set with prime numbers b_1, b_2, $b_2 > b_1$, 2D space is filled more uniformly than in the pseudo-random sampling case. Due to its low discrepancy and the fact that, unlike, e.g. stratified or jittered sample sets, Halton point sequences can easily be generated incrementally, the Halton sequences for primes $b_1 = 2$ and $b_2 = 3$ (Figure 3.6) are computed for comparative purposes.

It is clear from the illustration that whilst the quasi-random sequence is more uniformly distributed than the pseudo-random sample set, its members may be placed rather closely to each other. This effect is avoided by the uniform FastFPS point set reflecting its guaranteed bounds on intersample distances. Note that although the distribution of Halton point sets varies with the particular set of prime bases b_1, b_2 chosen, the problem of closely placed samples remains [172].

As regards the irregularity of the uniform FastFPS sample pattern, it is induced by the random initialisation of the farthest point sequence. This is reflected by the blue noise property of its power spectrum (Figure 3.7): A low energy disc around the zero frequency component with most of the power concentrated beyond a threshold frequency, i.e. aliasing is replaced by high-frequency white noise, so-called blue noise, indicating good anti-aliasing properties and the usefulness for a wide range of further processing tasks [34, 43, 107, 160, 161].

Thus, FastFPS of planar domains generates irregularly uniform sample patterns. It therefore produces sample sets of similar quality as the uniform image sampling technique of Eldar et al. [47].

FastFPS, however, extends the farthest point sampling principle more naturally to the adaptive sampling case. This is of importance since by varying the sampling rate with

[2]The pseudo-random sequences were computed with the help of algorithm "ran1" in Press et al. [131] which is reported to pass all established randomness tests for random number sequences of length less than 10^8.

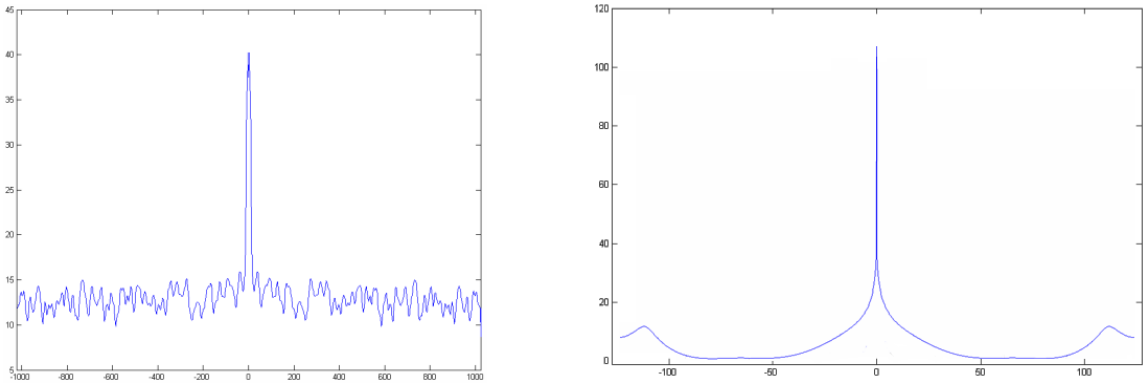

Figure 3.7: Power spectra [43, 161] (watts over frequency in Hz) of 1028 pseudo-random (*left*) and FastFPS samples (*right*). Whilst the energy distribution of the pseudo-random sequence is characterised by noise across all frequencies, i.e. white noise, the energy distribution of the FastFPS sequence displays typical blue noise properties: Most of the energy is concentrated beyond a frequency threshold, i.e. low-frequency aliasing is traded for high-frequency noise, which the human visual system is known to be less sensitive to than white noise or the false patterns characteristic of aliasing [134] (illustration is zoomed into a low frequency range to highlight this property).

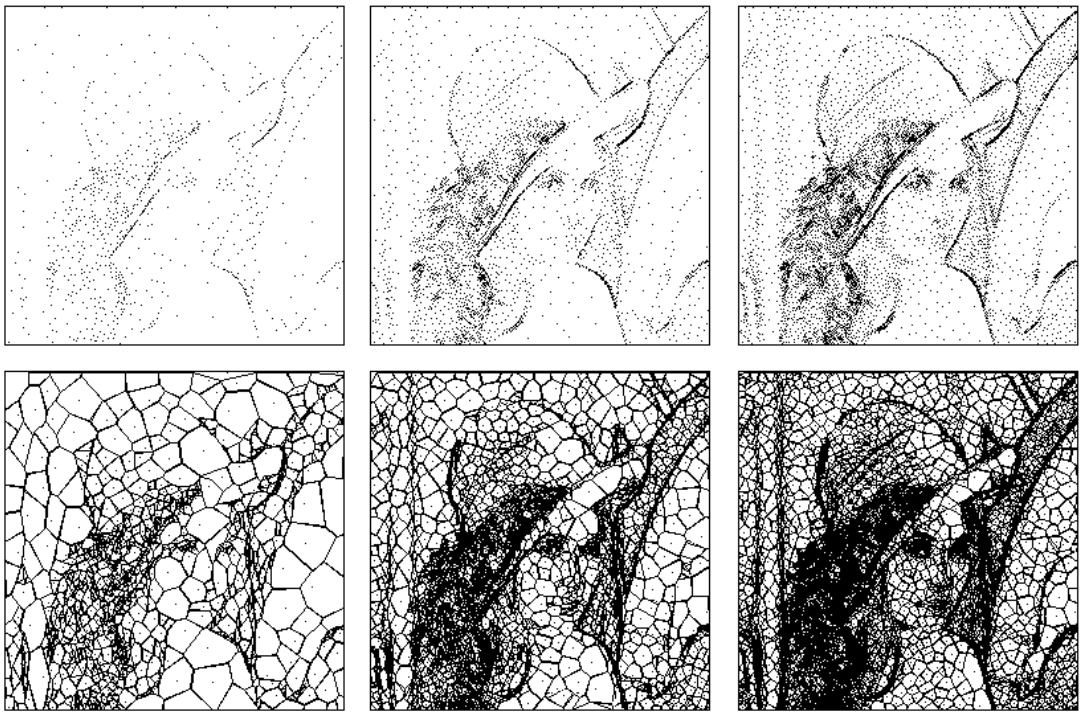

Figure 3.8: Adaptive FastFPS extends the farthest point sampling principle naturally by modelling discrete geodesic Voronoi diagrams in a metric reflecting the local image structure with the help of weighted intrinsic distance mapping. This is illustrated here by the discrete geodesic Voronoi diagrams (*bottom*) corresponding to the Lena sample sets of Figure 3.9 (*top*).

non-uniform image features, the number of samples required to meet the given sampling

criterion, e.g. the quality of the reconstructed image, can be reduced substantially [43]. In the FastFPS case, weighted intrinsic distance mapping across the domain models a tractable (discrete) geodesic Voronoi diagram in a metric reflecting the variation in an arbitrary local feature-sensitivity measure. This principle is illustrated in Figure 3.8. By choosing the vertex farthest away in this diagram, farthest point sampling is extended to the adaptive case without the need for experimentally determined weight functions incorporating a distance factor for the vertices in the Euclidean Voronoi diagram [46, 47]. Also, whilst Eldar et al. [47] adaptive image sampling by local bandwidth estimation represents a reasonable approach, in general, other sampling criteria need to be supported as well to optimise the sampling distribution for specific applications.

The flexibility of the proposed approach is shown in the following section.

3.6.4 Experimental results

To illustrate its flexibility, I use adaptive FastFPS for the generation of samples whose distribution follows an importance measure either given in form of a pre-computed importance map or computed on-the-fly during front propagation. The generation of such sampling patterns is required for numerous computer graphics applications such as rendering [55] or digital halftoning [160]. The section is concluded with experimental results on the computational efficiency of FastFPS of planar domains.

I use both a Fast Fourier Transform-based (FFT)[3] importance map of peak frequencies and CIELAB 1976-based colour-space [102] similarity measures. Colour dissimilarity is measured on-the-fly at each grid vertex either in the form of its Euclidean distance in CIELAB colour space from its 4-connectivity neighbourhood or using the CIELAB-based GLAB dissimilarity measure [61]. In either case, the largest distance is taken as the propagation speed at the vertex during simultaneous (line 2 in Algorithm 4) and partial weighted intrinsic distance mapping (line 9 in Algorithm 4). This way the distances between Voronoi vertices and the existing sample set in a relatively variable region are higher than they would be in a smoother area. The resulting sample distributions for a number of test images and sample budgets are shown in Figure 3.9. The images were chosen either for their colour dynamics (Monarch, Mandrill) or more subtle colour changes (Mona Lisa). The Lena image does not exhibit such properties but has developed into a standard test case over the years. Generally, the Mandrill image represents the most challenging test case due to the noisy nature of its fur.

[3] I use Don Cross' [36] FFT implementation. His code for the detection of the peak power frequency in the output of his FFT algorithm is also employed. The importance map consists of these peak frequencies.

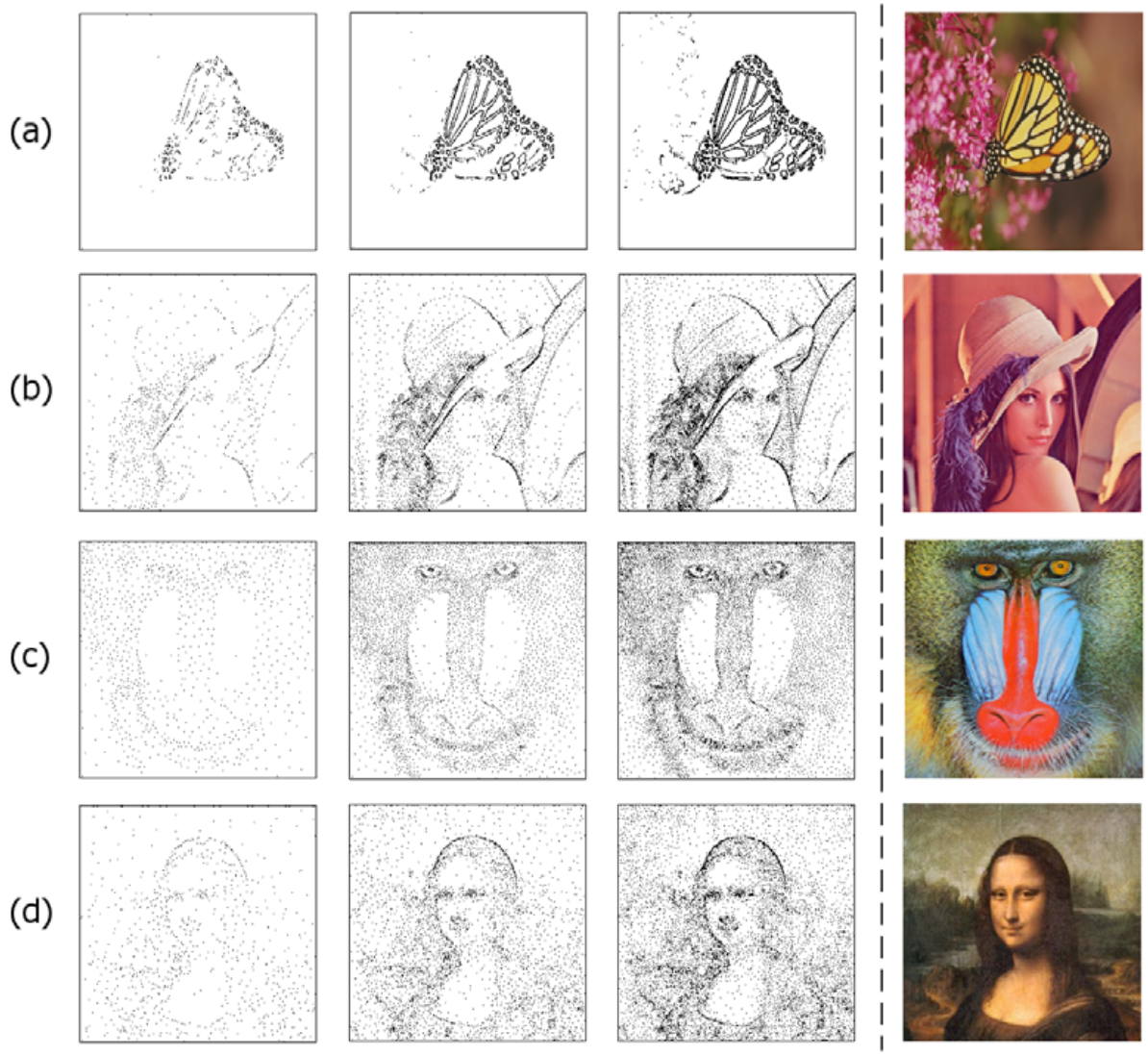

Figure 3.9: Adaptive FastFPS extends the farthest point sampling principle flexibly by allowing for (positive) weight functions reflecting local changes in arbitrary properties. This is illustrated here with FastFPS sample distributions obtained from FFT importance map (a), CIELAB colour space distance (b)(c) and GLAB similarity [61] (d) weight functions. The point sets consist of, from left to right, 1024, 4096 and 8192 samples. The (512×512) reference images are shown on the right.

Unsurprisingly, the peak frequency-based weight function leads to the strong concentration of samples near edges at the expense of the "adequate" sampling of smoother regions. For example, see Figure 3.9(a), where the smoothly varying background is allocated no samples at all. The sample distributions driven by the CIELAB colour space-based weight functions also capture important features early on into the sequence but avoid excessive over- or undersampling. In each case, the adaptivity of the sample distribution to the measured feature is clearly visible with the GLAB-based sampling of the Mona Lisa image perhaps yielding the most comprehensive representation for each sample budget. The

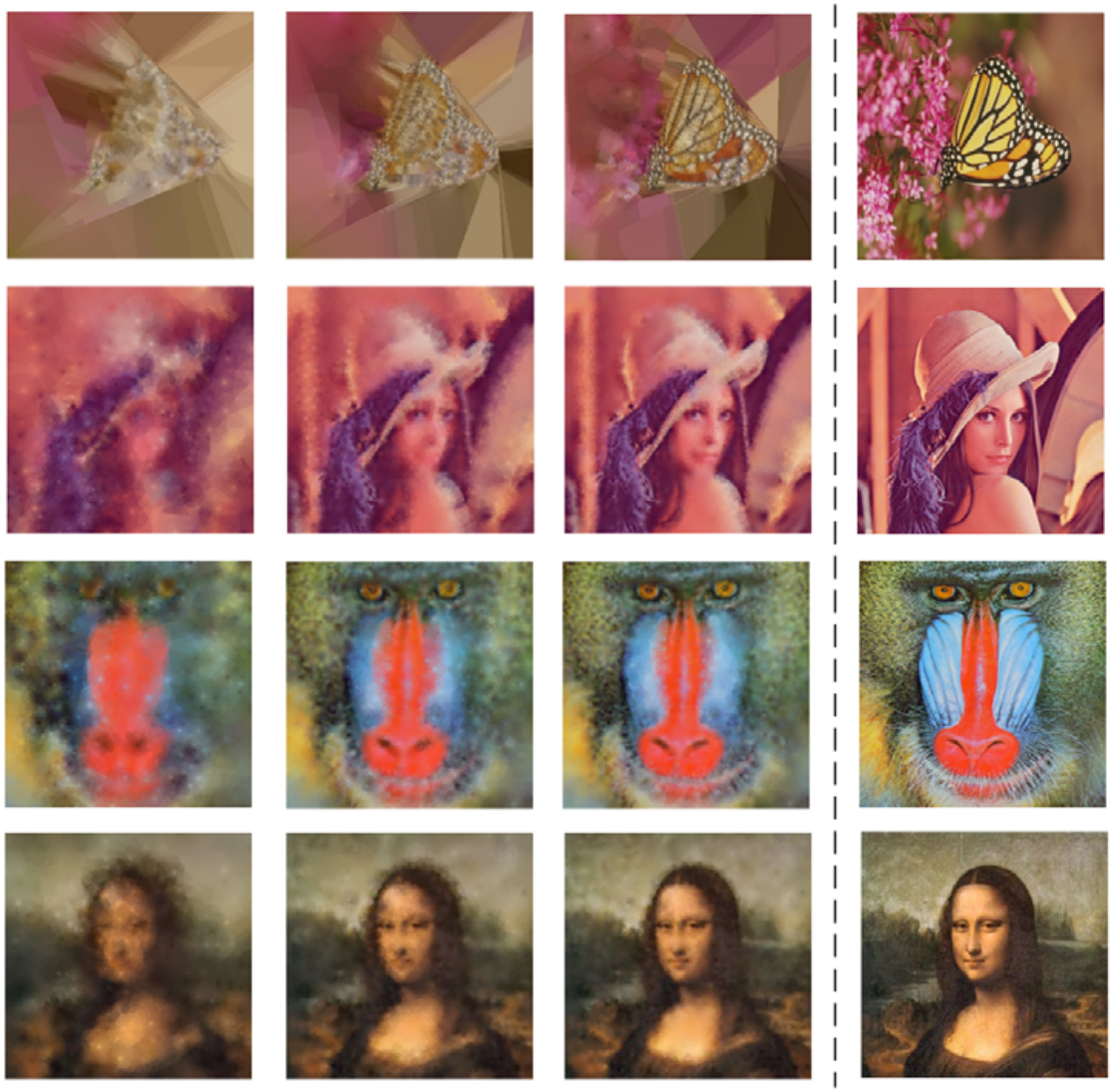

Figure 3.10: 4 nearest neighbour reconstructions corresponding to the sample sets of Figure 3.9. The reference images are again shown on the right.

CIELAB-based sampling of the Mandrill image deals very well with the noisy fur but fails to capture facial high-frequency detail adequately.

I follow Eldar et al. [47] and reconstruct the images using weighted 4 nearest neighbour interpolation: The intensity at a pixel q is computed as the average intensity at its four nearest sample points weighted by their reciprocal Euclidean distance from q,

$$I(q) = \frac{\sum_{i=1}^{4} I(s_i)/d(q, s_i)}{\sum_{i=1}^{4} 1.0/d(q, s_i)}, \tag{3.3}$$

where $s_i \in NN_q$, with $\|NN_q\| = 4$. Figure 3.10 shows the resulting reconstructions corresponding to the sample sets in Figure 3.9. I compute the peak signal-to-noise ratios ($PSNR$) as a measure of image distortion relative to the reference images. More

	1024	4096	8192
Monarch	13.63	14.32	15.41
Lena	16.64	20.15	22.67
Mandrill	15.41	17.85	19.20
Mona Lisa	20.25	23.91	26.10

Table 3.1: *PSNR* values for reconstructed images of Figure 3.10.

appropriate measures of *visual distortion* are reviewed in, for example, Taubman and Marcellin [155]. The *PSNR* is defined as

$$PSNR = 10.0 \times \log_{10} \frac{2^b \times 2^b}{MSE},$$

where b denotes the number of bits for the image intensity and *MSE* represents the mean-square reconstruction error,

$$MSE = 1.0/(uv) \sum_{y=1}^{v} \sum_{x=1}^{u} (I(x,y) - \hat{I}(x,y))^2,$$

for a $(u \times v)$ image; $\hat{I}(x,y)$ is the reconstructed pixel intensity in row x and column y. Since we are dealing with RGB colour images, I compute the *MSE* and *PSNR* values for each RGB colour channel separately, i.e. the intensity values range from 0 to 255 and thus $b = 8$ in each case, and average the results. The corresponding *PSNR* values for the reconstructed images are given in Table 3.1. Unsurprisingly, the reconstructions from sample sets generated with the help of the CIELAB colour space-based weight functions show higher *PSNR* values than the reconstruction of the Monarch image from the FFT-driven sampling and the reconstructions of the less challenging images (Lena, Mona Lisa) show the highest *PSNR* values throughout. For the most challenging test case (Mandrill), the noisy nature of the fur is captured relatively well but high-frequency detail is not recovered adequately. To achieve *PSNR* results typical of, for example, state-of-the-art image compression algorithms ($\geq$ 30dB), an (expected) reconstruction error-driven weight function and a more sophisticated reconstruction method would need to be considered. See, for example, Demaret et al. [39] (and references therein). Since the purpose of these application examples is to illustrate the flexibility of adaptive FastFPS, the pros and cons of the various feature-sensitivity measures, reconstruction methods and image or visual distortion measures are not considered any further.

To give an indication of the speed of planar FastFPS sampling, Table 3.2 provides execu-

	10k	20k	40k	80k	160k
(128×128)	0.31	0.60	-	-	-
(256×256)	0.42	0.77	1.44	-	-
(384×384)	0.63	0.94	1.63	3.07	-
(512×512)	0.94	1.26	2.02	3.71	6.97
(512×512)	2.82	6.16	10.54	19.42	39.06

Table 3.2: Execution times (in secs.) for the uniform FastFPS sampling of differently-sized versions of the Lena image for budgets of up to 160k samples. For comparative purposes, execution times of my implementation of Eldar et al. [47] uniform farthest point sampling algorithm are given in the bottom row.

tion times for the uniform sampling of differently-sized versions of the Lena image given different sample budgets.[4] Execution times for adaptive FastFPS vary with the nature of the weight function and whether or not the weights are computed on-the-fly or given in the form of an importance map. They are therefore not particularly useful as an indicator for the algorithm's execution efficiency and are not given here. Note, however, that in the case of adaptive FastFPS sampling using importance maps, Table 3.2 is indicative of the performance of adaptive FastFPS as well.

The graphical depiction of the results of Table 3.2 in Figure 3.11 shows that for a given sample budget, the execution time of FastFPS grows at worst linearly in the input size. This is considerably more efficient than the theoretical computational complexity of $O(N^2 \log N)$, where N represents the input size. Similarly, for a given input size, the execution time of FastFPS is slightly worse than linear in the output size. As documented in Table 3.2, the execution times of FastFPS of planar domains compares favourably with the execution times of my implementation[5] of Eldar et al. [47] uniform farthest point sampling algorithm.

In the following section, I turn to the sampling of geometry and present the instantiation of my generic sampling algorithm for triangulated surfaces.

[4] All computations were performed on a Pentium 4 2.8GHz, 512MB machine running under MS Windows XP.

[5] Eldar et al. [47] uniform image sampling algorithm was implemented with the help of the planar Voronoi diagram functionality of CGAL-2.4 [27].

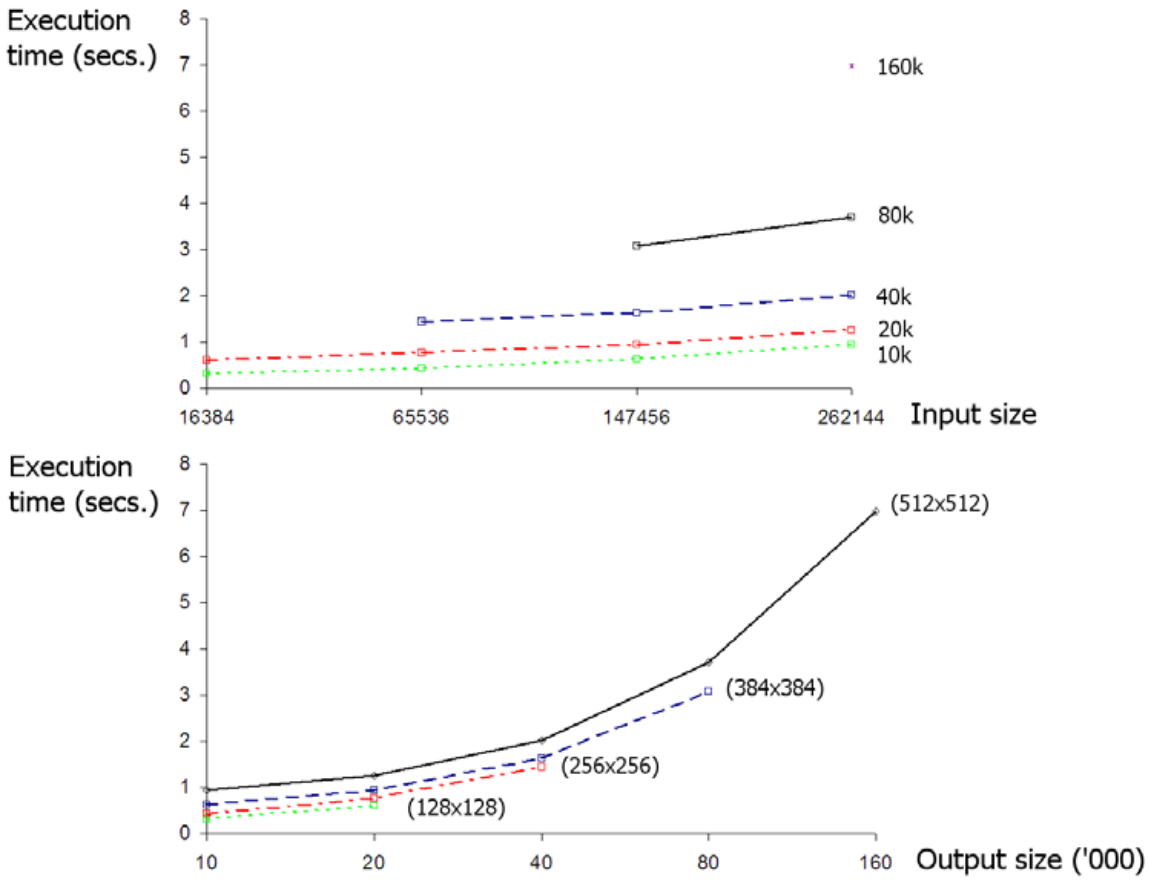

Figure 3.11: Two different graphical depictions of the FastFPS execution times of Table 3.2. The top figure shows the execution times for different sample budgets as a function of the number of input pixels. The bottom figure presents the results as a function of the output size given images of different sizes. The execution time of uniform FastFPS is at worst linear in the input size and slightly worse than linear in the output size.

3.7 FastFPS of triangulated surfaces

This section considers the instantiation of my point sampling algorithm template for triangulated surfaces.[6] Since this thesis is predominantly concerned with the intrinsic processing of point-sampled geometry and given the maturity of the most relevant application areas such as mesh simplification and remeshing, I focus on the description of the instantiation (Section 3.7.1) using my implementation of Fast Marching for triangulated surfaces, the instantiation's computational complexity (Section 3.7.2) and application examples (Section 3.7.3).

[6] A similar method was independently and simultaneously developed by Peyré and Cohen [127].

Input: Acute triangulation of P. Boundary (propagation source) vertex $q \in P$. Speed function $F > 0$.
Output: Weighted geodesic distance map of q.

---

0 *** *Initialisation* ***
1 Insert q in ALIVE with arrival time 0;
2 Insert in CLOSE, all triangle vertices edge-adjacent to q;
3 Initialise the points in CLOSE using a gradient approximation such as (2.4);
4 Insert all other triangle vertices in FAR with initial arrival times of "∞";
5
6 *** *Front propagation* ***
7 **REPEAT**
8 Let TRIAL denote the vertex in CLOSE featuring the smallest arrival time; Remove TRIAL from CLOSE and insert it in ALIVE;
9 Move all edge-adjacent neighbours of TRIAL which are FAR to CLOSE;
10 Using a gradient approximation such as (2.4), update the T-values of all CLOSE neighbours of TRIAL using only ALIVE vertices in the computation;
11 **UNTIL** all vertices are ALIVE;

Alg. 5: Fast Marching algorithm for triangulated surfaces in pseudocode.

3.7.1 Implementation of the generic algorithm for triangulated surfaces

I start with the discussion of my implementation of Fast Marching for triangulated surfaces (reproduced in Algorithm 5). This module replaces the Cartesian Fast Marching module used in the case of FastFPS of planar domains. This is followed by the discussion of the template instantiation for triangular meshes.

The key differences of Fast Marching for triangulated surfaces from Cartesian Fast Marching include the need for both a mesh as opposed to a structured grid data structure in the grid module and a mesh-based arrival time update procedure as well as an approach towards dealing with obtuse triangles. The other processing steps of Algorithm 5 are analogous to the Cartesian case and are therefore not discussed any further.

I use the half-edge data structure of CGAL-2.4 [27], described in detail in Kettner [82], as mesh data structure for the representation of the triangulated surfaces. For each vertex, I store its incident half-edge and this half-edge only, i.e. to improve memory efficiency, no face information is stored since it is of no further use in the context of triangulated Fast Marching. Each vertex further holds its indices in a min- and a max-heap used for Fast Marching support and farthest point sample candidate retrieval respectively. This completes the unstructured grid module implementation.

The implementation of the gradient approximation underlying triangulated Fast Marching

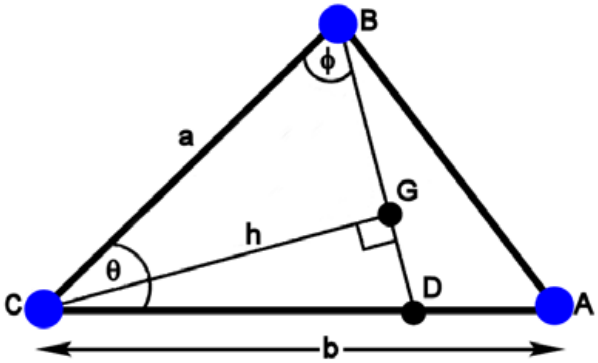

Figure 3.12: Terms used in update scheme of Fast Marching for triangulated surfaces.

needs to distinguish between the following cases during the initialisation (line 3) and updating phases (line 10). If both $T(A)$ and $T(B)$ are FAR, it is clear that the arrival time of vertex C will not be updated from $\triangle ABC$ and the next triangle sharing vertex C can be considered. In case $T(A)$ is ALIVE and $T(B)$ is FAR, $T(C)$ gets updated along edge AC, with length b, only (Figure 3.12), i.e. $T(C) = \min\{T(C), T(A) + bF(C)\}$. Similarly, if $T(A)$ is FAR and $T(B)$ is ALIVE, $T(C)$ follows as $T(C) = \min\{T(C), T(B) + aF(C)\}$. If both $T(A)$ and $T(B)$ are ALIVE, the quadratic equation (2.4) needs to be solved for its largest root. If its discriminant is non-negative and the solution for t in (2.4) satisfies the condition $t > u$, where $u = T(B) - T(A)$,[7] t represents a valid solution and the new estimate of $T(C)$ is found as $T(C) = T(A) + t$; otherwise, or in case the condition of $T(C)$ being updated from within $\triangle ABC$, i.e. monotonicity condition (2.5), is violated, the new $T(C)$ estimate follows as the minimum update in the edge directions, i.e. $T(C) = \min\{T(C), T(A)+bF(C), T(B)+aF(C)\}$. This procedure is repeated for each triangle sharing C. The smallest of these estimates is then chosen as the new estimate for $T(C)$.

The above procedure already indicates the approach towards dealing with non-acute triangulations. In the case of triangulations featuring obtuse triangles, even if condition (2.5) is met, monotonicity may not hold, i.e. the gradient of the solution may not point into the triangle from which the solution was computed. This is not the case when dealing with acute triangulations. However, rather than adding a non-trivial pre-processing step in the form of recursive triangle unfolding to identify a second "virtual" vertex to modify an obtuse into two acute triangles, I follow Adi and Kimmel [2] and compute the arrival time estimate as the minimum of the one known and the one estimated arrival time in the grid directions, i.e. $T(C) = \min\{T(C), T(A) + bF(C), T(B) + aF(C)\}$. Computing $T(C)$ on the basis of this partial information affects the accuracy of the gradient approxima-

[7] Without loss of generality, it is assumed that $T(B) > T(A)$.

tion [146]. However, the application examples of Section 3.7.3 indicate that this represents a satisfactory solution for the test cases used, most of which feature largely well-behaved triangulations. For triangulations with strongly obtuse triangles, an alternative method needs to be used, see, e.g. Sethian and Vladimirksy [146].

Arrangement of the edge-adjacent neighbours of an ALIVE vertex in a narrow band and the optimal sorting of their arrival time estimates in a min-heap yields again a Fast Marching module of $O(N \log N)$ complexity, with N denoting the number of triangle vertices [84]. This module is used to implement FastFPS of triangulated surfaces as described in the following.

With the Fast Marching and grid modules in place, the remaining implementation is analogous to the planar domain case. That is, processing starts with the selection of the first sample vertex, s_1, at random; s_2 and s_3 follow as the vertices farthest away after computation of $T_M(s_1)$ and $T_M(s_2)$ respectively using the Fast Marching module. Simultaneous propagation of circular fronts from s_1, s_2, s_3 outwards then yields the discretised bounded $VD(S)$ and therefore the first set of farthest point candidates in the form of the vertices of $VD(S)$. These vertices are stored in a priority queue.

This priority queue is again implemented in the form of a max-heap with its elements sorted in descending order by their arrival times. Thus, the next farthest point candidate is located at the root. Re-heapification is implemented with the help of the method's friendly access to the grid module. This way the max-heap index stored at a triangle vertex can be updated with its new value in constant time following root extraction, vertex removal, or vertex insertion. Thus, the heap module remains unchanged.

Partial weighted intrinsic distance mapping is performed by propagating a front locally from s_{n+1} outwards until loci of equal arrival time values are encountered. This yields bounded $R(s_{n+1}, S')$ and thus $VD(S')$. Voronoi vertices which have become obsolete due to the expansion of $R(s_{n+1}, S')$ into their regions are deleted from the max-heap using their max-heap indices stored at the relevant triangle vertices. The Voronoi vertices of $R(s_{n+1}, S')$ are added to the max-heap.

The refinement condition ρ is enforced by controlling for the arrival time of the max-heap root being greater than or equal to ρ.

Hence, apart from the use of an unstructured grid and a different Fast Marching module, the implementations of FastFPS of planar domains and FastFPS of triangulated surfaces coincide. The algorithm's computational complexity is discussed next.

3.7.2 Computational complexity

Compared with the instantiation of template Algorithm 4 for the planar domain case (Section 3.6.1), only the Fast Marching module and the underlying grid data structure have to be replaced. The traversal of the half-edge data structure does not add significantly to the computational complexity. The new Fast Marching module retains the running time of $O(N \log N)$, N representing the number of triangle vertices. The remaining implementation modules are either unchanged (heap module) or are of similar computational complexity to the planar domain case (weight function module). Thus, the overall computational complexity of FastFPS for triangulated surfaces is $O(N^2 \log N)$.

I give a number of typical application examples for FastFPS of triangulated surfaces in the following section.

3.7.3 Experimental results

In the following, I highlight a number of applications for FastFPS of triangulated surfaces including, in particular, uniform and non-uniform mesh simplification and continuous level-of-detail generation. The presentation starts with a brief qualitative look at the sample distributions produced by this particular instantiation of Algorithm 4.[8]

To indicate the nature of the sample distributions generated by uniform FastFPS of triangulated surfaces, the Venus model is sampled for different refinement thresholds. The refinement thresholds are set as a percentage of the model's bounding volume diagonal. The resulting geometry is shown without any mesh connectivity in Figure 3.13. It can be seen that the sample vertices are distributed irregularly uniformly without any clustering or large holes. As in the planar domain case, this is the result of the algorithm's guaranteed deterministic min-max intersample distances and its random initialisation.

The corresponding sample sets are shown with their mesh connectivity in Figure 3.14. These results represent examples for the use of FastFPS of triangulated surfaces for the intrinsic (uniform) subsampling, i.e. the intrinsic simplification, of a triangular mesh subject to minimum density conditions.

To intrinsically subsample a triangular mesh feature-sensitively, adaptive FastFPS of triangulated surfaces is used with a weight function computed on-the-fly. The function captures the mean surface curvature. Alternatively, the sampling could be driven by

[8] All computations were performed on a Pentium 4 2.8 GHz, 512MB machine running under Windows XP.

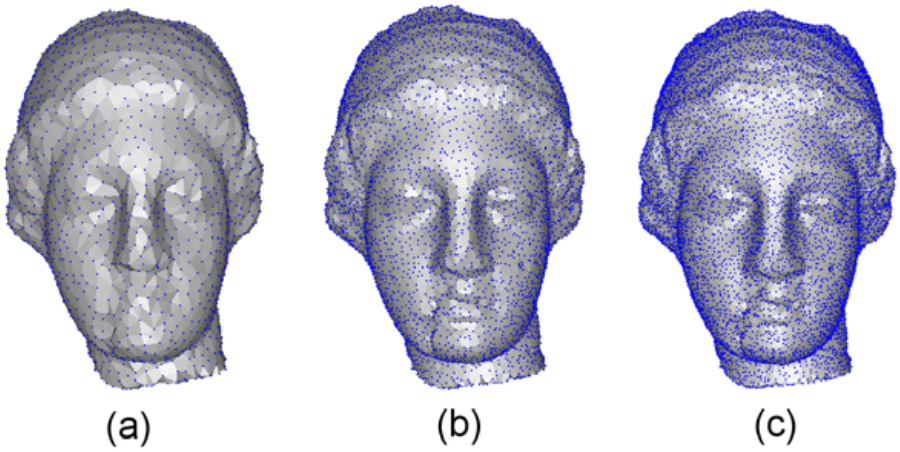

Figure 3.13: Uniform FastFPS of the Venus model (134345 vertices). The model is intrinsically simplified subject to refinement conditions of 6.0% (a), 4.0% (b) and 1.0% (c) of the model's bounding volume diagonal. The sample sets are of size 2980 (a), 4842 (b) and 19866 (c) and are shown without mesh connectivity to indicate the cluster- and hole-free quality of the sample sets.

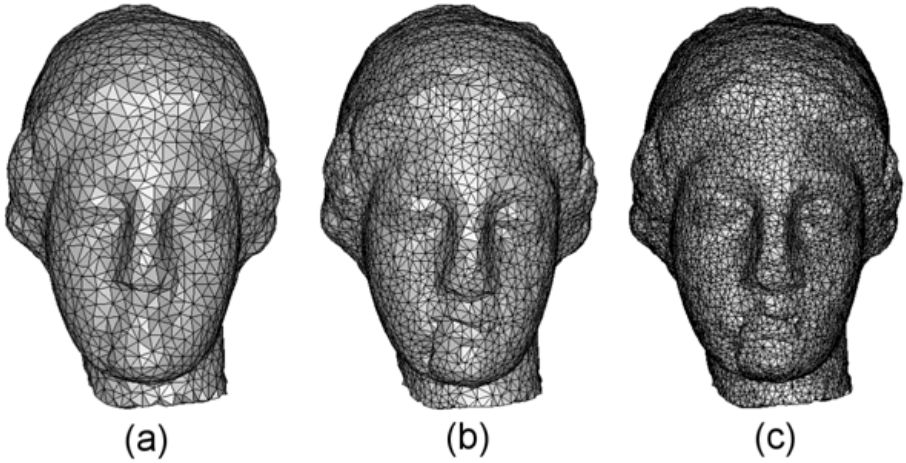

Figure 3.14: Uniform FastFPS of the Venus model. The sample sets of Figure 3.13 are shown with their mesh connectivity. These results represent examples for the use of the algorithm for intrinsic mesh simplification.

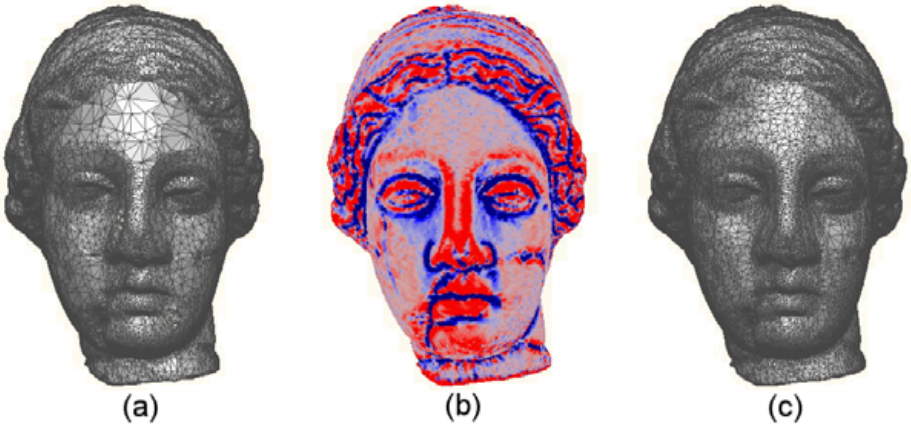

Figure 3.15: Adaptive FastFPS of the Venus model driven by mean curvature estimation. The sample distributions of the 20.0% (26870 vertices) (a) and 30.0% (40300) (c) intrinsically simplified models follow the surface's mean curvature plot (b). As a result, features are covered by samples much more strongly than in the uniform case (Figure 3.14). In (b), regions of strong curvature are illustrated in red, blue indicates low curvature.

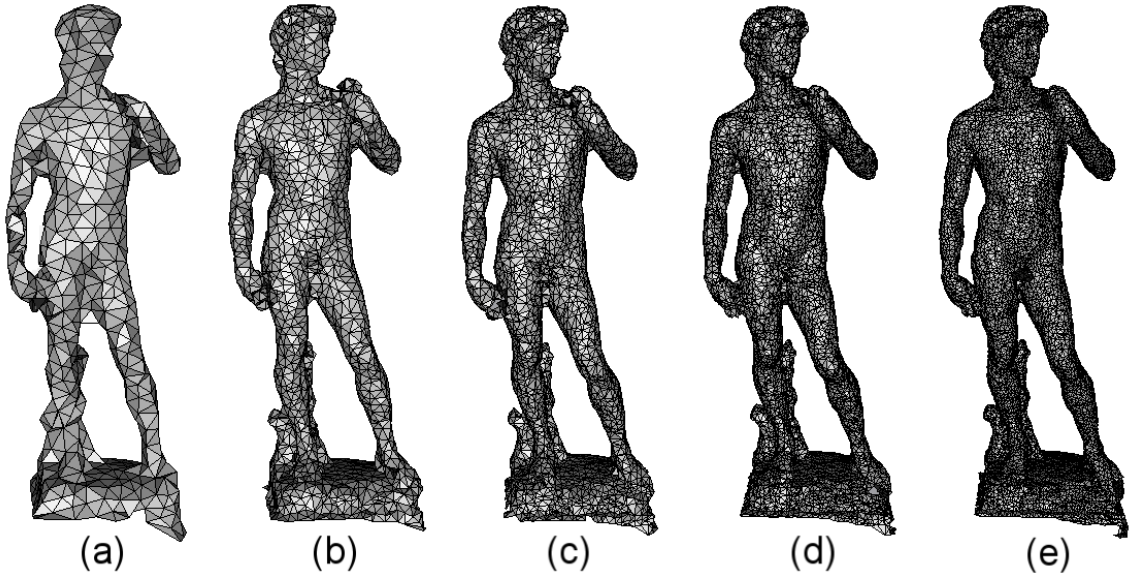

Figure 3.16: Uniform FastFPS of the Michelangelo David model (1 million vertices). The coarse-to-fine, progressive nature of FastFPS is exploited for the continuous generation of level-of-details. The individual level-of-details shown here consist of 700 (a), 2500 (b), 5000 (c), 10000 (d) and 20000 (e) vertices respectively.

changes in other surface features such as texture intensities or the combination of different feature or view-dependency measures. A surface's mean curvature is measured here by computing the divergence between normals estimated at both the vertex under consideration and its edge-adjacent vertices. Since the faces of the triangular mesh are planar, each face has a well-defined normal vector. The normal vector n_{q_i} at a vertex q_i is estimated as the normalised weighted sum of the normals of the faces incident to q_i. The weights are chosen proportionally to the surface areas of the faces, i.e.

$$n_{q_i} = \frac{\sum_{f_j \in F_{q_i}} \|f_j\| n_{f_j}}{\| \sum_{f_j \in F_{q_i}} \|f_j\| n_{f_j} \|},$$

where F_{q_i} represents the set of faces f_j of the triangles sharing vertex q_i; $\|f_j\|$ denotes the area of face f_j and n_{f_j} represents its (unit) normal vector [154]. The mean curvature is then approximated by estimating the change between the estimated normal at the vertex under consideration and the estimated normals at its edge-adjacent neighbours. This estimate is computed with the help of Wild Magic 2.4 [170]. The relevant library method computes the derivative matrix of the estimated local normal vector field and takes the trace of the matrix as the mean curvature estimate. The results produced by adaptive FastFPS of the Venus model using this mean curvature-based weight function are given in Figure 3.15. Figure 3.15 also shows the mean curvature plot of the non-simplified model generated using RapidForm 2004 [132]. Relative to the uniform sampling shown in Figure

3.14 in particular, the FastFPS samples are more strongly concentrated in regions of strong (mean) curvature. Adaptive sampling may again also be driven using an importance map, for example, in the form of a texture image.

Finally, uniform FastFPS suggests itself for the continuous generation of level-of-details of a model. The generation of level-of-detail representations is of interest in the context of applications such as progressive model transmission across limited bandwidth channels or selectively-refined rendering [71]. Figure 3.16 presents various level-of-details of the Michelangelo David model generated by uniform FastFPS of triangulated surfaces for different sample budgets. Uniform FastFPS suggests itself for this purpose due to its progressive, coarse-to-fine nature: Once a subsampled base model consisting of a given minimum number of vertices has been computed, the level-of-detail sequence can be generated continuously by continuously and losslessly selecting new farthest point samples until the full resolution model is obtained.

This completes the presentation of application examples for FastFPS of triangulated surfaces. The following section summarises and discusses the results of this chapter.

3.8 Summary and discussion

In this chapter, the (Euclidean) farthest point sampling algorithm introduced by Eldar et al. [47] was generalised to an algorithm template for the intrinsic uniform and adaptive sampling of all geometry representations for which the notion of geodesic Voronoi diagram is well-defined. That is, unlike other farthest point sampling algorithms [21, 29, 47, 127, 128, 148], it is applicable to both planar domains and triangulated surfaces as well as surfaces in point cloud and implicit form and thus supports the most widely-used forms of geometry representation. The algorithm operates intrinsically and thus geometry-sensitively throughout. The generated sample patterns observe deterministic min-max bounds guaranteeing favourable domain coverage and the support of meaningful further processing. Unlike the algorithm of Eldar et al. [47], my intrinsic farthest point sampling principle extends naturally and flexibly to the adaptive sampling case. The user thus can control both the density of the sampling and its uniformity. The algorithm design is modular. Depending on the target surface representation, a different Fast Marching module and grid representation needs to be plugged in but the algorithmic structure of the template remains unchanged. By keeping the weight function module separate from the other modules of the implementation, the algorithm can be adjusted to arbitrary sensitivity criteria by simply replacing this module. Thus, we have a toolbox for the

custom-tailored intrinsic sampling of planar domains and manifold surface representations.

The main contributions of this approach may be summarised as follows

- Generality - The algorithm template applies to images and geometry in the most widely-used forms of representation.
- Modular design - Template instantiation for different surface representations only requires the replacement of the Fast Marching and the grid representation modules.
- Sampling density guarantee - The algorithm guarantees intersample geodesic distances bounded deterministically in a min-max sense.
- Flexibility - The algorithm extends the farthest point sampling principle naturally to the adaptive case allowing for pre-computed or on-the-fly weight functions of arbitrary properties.

It should be noted, however, that the sampling density guarantee assumes the computation of continuous geodesic Voronoi diagrams. Since my implementations discretise the geodesic Voronoi diagram computation, the sampling density guarantee is affected by discretisation error in practice. The experimental results do not suggest this to be significantly detrimental to the quality of the sample distributions.

The algorithm template was instantiated for the case of planar domains with the help of 2D Cartesian Fast Marching. The main strengths of the resulting FastFPS technique for planar domains follow the strengths of the algorithm template and consist of

- Flexibility - Subject to a positive propagation speed throughout, FastFPS of planar domains may be driven by pre-computed or on-the-fly weight functions of arbitrary properties.
- Efficiency - FastFPS of planar domains performs significantly better in practice than suggested by its $O(N^2 \log N)$ theoretical computational complexity.
- Blue noise - FastFPS of planar domains generates sample distributions featuring blue noise power spectra.

Overall, the favourable properties of the original uniform farthest point image sampling algorithm [47] have been retained. In addition, however, FastFPS of planar domains shows favourable execution efficiency in practice and allows for more flexible adaptive sampling.

The use of weighted intrinsic distance mapping implies that FastFPS visits all pixels of an image, possibly multiple times. This is in contrast to typical image sampling algorithms such as Eldar et al. [47] which analyse the image at existing sample sites only to support image acquisition/compression and/or to improve execution efficiency. By contrast, FastFPS of planar domains assumes that the image is accessible in its entirety. The experimental results have shown that FastFPS performs competitively nevertheless, for example, outperforming my implementation of the algorithm of Eldar et al. [47] (Table 3.2).

The algorithm template was instantiated for the triangulated surface case using my implementation of Fast Marching for triangulated surfaces. Apart from this Fast Marching module and the unstructured grid data structure, the modules of FastFPS of planar domains can be re-used. The strengths of the resulting FastFPS algorithm are similar to the planar domain case and consist of efficiency, favourable sample distributions of guaranteed density and flexible adaptive surface sampling. For example, apart from geometric measures such as local curvature or triangle shape, similarity of colour associated with vertices or triangles may be considered. In addition, due to its intrinsic nature, the algorithm operates inherently sensitively to the geometry of the triangular mesh. I briefly indicated typical applications of FastFPS of triangulated surfaces in the form of intrinsic uniform and adaptive mesh simplification and continuous level-of-detail generation.

The lack of a guaranteed approximation error between the original and the simplified model represents the main weakness of the use of the algorithm for a purpose such as mesh simplification. However, compared with iterative fine-to-coarse simplification algorithms [31, 58], FastFPS of triangulated surfaces does not require the re-evaluation of the simplification criterion in the neighbourhood of the last simplification operation every time a simplification step has been performed. The computationally most demanding processing step of iterative fine-to-coarse simplification algorithms is thus avoided [58]. Note in this context that most existing mesh simplification algorithms do not provide meaningful guarantees on the quality of the generated approximation of the original mesh. Those methods which do give such guarantees tend to be inefficient and do not necessarily produce better results [31]. Nevertheless, guaranteed approximation error thresholds are frequently required in application contexts such as medical imaging or finite element methods.

Since this thesis is primarily concerned with the processing of surfaces in point cloud form, a more detailed comparative analysis of the strengths and weaknesses of FastFPS of triangulated surfaces for certain applications is beyond its scope.

Peyré and Cohen [127] developed a sampling algorithm for triangular meshes conceptually similar to FastFPS of triangulated surfaces. The authors, however, fail to take advantage of the relationship between incremental geodesic Voronoi diagram construction and the generation of an intrinsic farthest point sequence. As a result, rather than maintaining a priority queue of Voronoi vertices, i.e. farthest point candidates, for fast sample point retrieval, Peyré and Cohen [127] opt for the less efficient alternative of searching for the mesh vertex with the largest distance map value each time a new farthest point sample is required.

In the following chapter, I turn to the use of my intrinsic farthest point sampling concept in the context of point-sampled geometry processing.

Chapter 4

Intrinsic meshless surface simplification

Modern 3D data acquisition devices produce surface representations in the form of point sets of substantial density due to submillimeter measurement precision. Surface reconstruction algorithms [8–10, 15, 18, 19, 38, 45, 72] either fail to cope with the inherent redundancy of these point sets or produce highly dense surface meshes. To facilitate meaningful further mesh-based processing, these meshes require mesh simplification algorithms [58] (and references therein) which are frequently as time and memory demanding as the preceding surface reconstruction step. By meshlessly simplifying the point set surface representation first and, if required, generating the surface mesh from the simplified point set, the surface reconstruction problem is accelerated significantly and the mesh simplification step is avoided altogether. Alternatively, surface reconstruction and thus mesh-based processing and its inherent maintenance overhead may be completely replaced by more efficient point-based modelling [5, 97, 125, 126, 178] and visualisation algorithms [22, 59, 76, 78, 95, 129, 139, 179, 180] at little or no loss in quality. In either case, the meshless simplification of the input point set represents a vital first processing step. Due to the prevalence of the approach, surface simplification is often understood to be a fine-to-coarse surface sampling process. Note, however, that the problem can be approached in either direction by sampling the surface representation from coarse to fine, which is the approach adopted here.

The problem addressed in this chapter may be stated formally as follows: Given a surface representation in the form of the set of points P, $\|P\| = N_1$, simplify P to a point set P' of target output size $N_2 < N_1$ subject to user-controlled refinement condition $\rho > 0$. The input point set does not have to be uniformly distributed. In the following, I propose

to address this problem intrinsically with the help of FastFPS for surfaces in point cloud form.

More specifically, the aim is to develop a dedicated point cloud simplification algorithm allowing for simple density control whilst satisfying a set of important requirements. These are a point set density guarantee to allow for meaningful further processing, memory and execution efficiency, and support for both uniform and user-designated feature-sensitive simplification. The algorithm should not be restricted to the generation of subsets of the input point cloud but support its resampling as well. Meeting this requirement would help to satisfy the final requirement of being able to deal meaningfully with non-uniformly distributed input point sets and illegitimate holes in the form of undersampled regions.

Section 4.1 reviews relevant related work. My intrinsic point cloud subsampling and resampling techniques are presented in Section 4.2 and Section 4.3 respectively. Section 4.4 provides application examples and experimental results for a number of massive data sets. Details of my Moving Least Squares (MLS) and enhanced k nearest neighbourhood implementations are given in Section 4.5. Section 4.6 summarises and discusses the results.

This chapter is based on Moenning and Dodgson [112, 113].

4.1 Related Work

The following review focuses on dedicated point cloud simplification algorithms, with an eye to how they perform against the set of requirements listed in the previous section.

Dey et al. [41] were among the first to present a dedicated point cloud simplification algorithm which exploits the particular structure of 3D Euclidean Voronoi regions of a densely distributed input point set both to detect oversampled regions and to determine candidate points for removal. Subsequent point decimation observes a user-controlled density condition. Due to the use of the medial axis-related local feature size concept [8] (Appendix B), their method is inherently sensitive to changes in local curvature. The algorithm does not support adaptive decimation driven by changes in any measure other than or in addition to local curvature. It is restricted to the generation of a subset of the input point set and cannot handle non-uniformly distributed point clouds or point sets featuring illegitimate holes. It requires the computation and maintenance of 3D Euclidean Voronoi diagrams and therefore tends to be computationally and memory demanding.

Linsen's [97] simplification method for point sets associates each input point with an information content measure and iteratively deletes points with lowest entropy. The in-

formation content measure represents a weighted sum of local curvature, non-uniformity and colour variation computed in the candidate point's k nearest neighbourhood enhanced by a maximum angle criterion, i.e. an enhanced k nearest neighbourhood (Section 2.4.2). The simplification algorithm is simple and effective but does not give any density guarantee and is limited to the generation of point cloud subsets. Input point clouds may therefore be simplified to prohibitively unevenly distributed point sets and non-uniformly distributed input point sets will typically result in non-uniformly distributed output point sets. The resampling of the input point cloud, which may be necessary in either case to support any effective further processing, is not addressed in the paper.

The point decimation scheme of Alexa et al. [4] judges each input point's importance by its distance from its projection onto the MLS surface (Section 4.3.1) computed from all input points other than the point under consideration. Those points exhibiting the smallest distance are considered redundant and are removed iteratively. Flexible feature-sensitive simplification is not supported. Similar to the techniques discussed above, this method produces a subset of the input point cloud and may require resampling to avoid any excessive non-uniformity in the simplified point set. Such a resampling method is not provided in the paper.

Pauly and Gross [123] present a resampling technique as part of their spectral processing pipeline for point-sampled geometry. Their algorithm tessellates the point cloud into a set of overlapping surface patches which are resampled on a regular grid. The discrete Fourier transform of this regular sampling yields the spectral representation of the surface patch. The patch can then be resampled with the availability of the Fourier representation allowing for the determination of the optimal sampling rate for the signal previously filtered according to a maximum approximation error. The simplification method itself is efficient and inherently sensitive to local curvature. The sampling pattern, however, varies with the chosen patch layout. Also, the (non-trivial) stitching back together of the resampled patches requires a blending function for the sampling density across patch boundaries. Although the algorithm features effective local error control, it is unclear how the global error is affected by the blending of sample densities across patch boundaries. It seems difficult to control the target output size through the spectral error bounds. For truly massive data sets, i.e. point clouds substantially larger than the largest point set of 4128614 points considered in the paper, the patch generation, regular resampling, discrete Fourier transform and spectral analysis pre-processing steps are prohibitively expensive and, consequently, out-of-core processing is left as future work. Also, non-uniform resampling by criteria other than local curvature is not supported.

Pauly et al. [124] adapt various widely-used mesh simplification techniques to the mesh-

less surface simplification scenario. Their iterative simplification method is reported to produce the best results in terms of average geometric accuracy but does not allow for simple control of point set density and requires expensive pre-computations. Particle simulation is found to represent the next-best method in terms of approximation accuracy and the best choice in terms of point set density control but is generally computationally demanding. Uniform incremental clustering is computationally efficient but is reported to produce the highest approximation error and is not naturally extensible to simplification sensitive to measures other than changes in local curvature. Similarly, hierarchical clustering is memory and execution efficient but even in its adaptive version yields point sets of approximation error only slightly lower than that introduced by the method performing poorest on this criterion, uniform incremental clustering. The methods support surface resampling but do not come with any sampling density guarantee.

In summary, while all of the simplification algorithms discussed above meet a subset of the requirements listed in the previous section, none satisfies them all. My approach towards meeting these requirements is presented next.

4.2 Intrinsic point cloud subsampling

The instantiation of algorithm template Algorithm 4 for surfaces in point cloud form yields an intrinsic meshless subsampling technique for surfaces represented by uniformly dense point clouds. The details of this instantiation are discussed in the following.

My implementation of Fast Marching for surfaces in point cloud form is discussed in Section 2.5.2. This implementation is used here as the model for the algorithm template's Fast Marching module. The model for the (structured) grid module is given by the grid mapping described in Section 2.5.1. This lookup table is computed in a one-off pre-processing step. The grid vertices which make up Ω_P^r, and those grid points only, for a fixed r and a given grid resolution are retained. All other vertices are discarded from the table to improve memory efficiency. The heap module implementation coincides with that used for Cartesian Fast Marching (Section 2.5.1). The actual subsampling process resembles the sampling algorithm for planar domains and triangulated surfaces discussed in the previous chapter and can be summarised as follows.

---

1. Initialisation: The algorithm starts by reading in the (pre-computed) offset grid and the embedded point cloud P and from it selects an initial sample point, s_1, at random. The grid vertices enclosing s_1 are initialised with their analytic distance from s_1. A sec-

ond, s_2, and third, s_3, starting sample are generated by repeatedly selecting the point farthest away in the weighted intrinsic distance maps of s_1 and s_1, s_2 respectively. Once $\|S\| \geq 3$, the algorithm constructs the initial discrete bounded geodesic Voronoi diagram, $VD(S)$, by simultaneously propagating fronts from the initial sample points outwards. The Voronoi vertices' arrival times are inserted into the max-heap data structure.

2. Sampling: The algorithm proceeds by extracting the root from the max-heap. This yields the next farthest point sample in the form of the input point closest to the root's grid location. This sample is inserted into $VD(S)$ by resetting its arrival time to zero and propagating a front away from it using partial weighted intrinsic distance mapping. This way, the front will continue propagating until it hits (grid) vertices featuring equal or lower arrival times and thus belonging to neighbouring Voronoi regions. The arrival times of updated grid vertices are updated correspondingly in the min-heap using back pointers. New and obsolete Voronoi vertices are inserted or removed from the max-heap respectively. The algorithm continues extracting the root from the max-heap until the sample point budget has been exhausted or the refinement condition has been met. This intrinsic point cloud subsampling technique is made adaptive by allowing the propagation speed F to vary with any (positive) point weights either computed on-the-fly or imported into the weight function module in the form of pre-computed importance maps.

The refinement condition is formulated in the form of the user-controlled density condition $\rho > 0$ of Section 3.3. That is, the simplified point set is refined until the next sampling candidate is closer in geodesic distance to the set of existing samples than ρ. The condition $\rho > 0$ ensures that the algorithm terminates. In particular, if ρ is positive but small, the process terminates when all the input points have been sampled. Alternatively, instead of setting a refinement threshold, the user may set the sample budget N_2.

The Fast Marching and heap module implementations retain a computational complexity of $O(N \log N)$ and $O(\log N)$ respectively, N denoting the number of grid vertices in Ω_P^r. Also note that the subsampling process coincides with that of FastFPS of planar domains. As a result, the algorithm's computational complexity is $O(N^2 \log N)$. The run-time observed in practice is again considerably more favourable as documented in Section 4.4 below. The algorithm's memory requirements benefit from the consideration and storage of grid vertices located in the offset band only. The actual requirements correspondingly vary proportionally with the size of the offset band.

This completes the discussion of the intrinsic point cloud subsampling algorithm which follows from the instantiation of Algorithm 4 for the point cloud case. As a result, we have

a coarse-to-fine, progressive algorithm for the intrinsic uniform and adaptive subsampling of points clouds with sampling density guarantee.

Point-sampled geometry can feature non-uniform point distributions and illegitimate holes. This is often the result of shortcomings of the data acquisition process such as inadequate lighting conditions, specular reflectance properties or occlusions in concave surface areas when using optical data acquisition techniques such as laser range scanning [141]. In these cases, subsampling unduly restricts the possible positions of sample points which can result in non-uniform sample distributions. It is therefore desirable to have a resampling method available which can be used to generate a more uniform sample distribution at the presence of local point distribution non-uniformities. Such a method is presented in the following section.

4.3 Intrinsic point cloud resampling

When dropping the assumption of uniformly distributed input point sets, pure subsampling is likely to produce output point sets unsuitable for meaningful further processing. This is due to the fact that the subsampling of non-uniformly distributed input point sets will typically result in non-uniformly distributed output point sets. The distribution non-uniformities of these point sets undermine operations such as surface editing or rendering. Thus, in the case of non-uniformly distributed input point sets, a resampling algorithm is needed which requires that a number of issues are addressed. Firstly, the resampling of a point cloud surface representation requires local surface approximation. Since this thesis is predominantly concerned with meshless surface processing, this is performed meshlessly using Alexa et al. [4] MLS method, summarised in Section 4.3.1. Apart from this additional MLS operation, steps (1) and (2) of the simplification algorithm remain unchanged. Secondly, the radii of the offset balls, $B(p_i, r_i)$, $p_i \in P$, can no longer be constant but need to be adjusted to account for local density variations and undersampled regions. This is discussed in Section 4.3.2. Finally, as already noted in Section 2.4.2, conventional point neighbourhoods are of limited usefulness in the presence of non-uniformities of the point distribution. For the determination of localised regression weights and non-constant offset ball radii, the enhanced k nearest neighbourhood concept (Section 2.4.2) is therefore used as also discussed in Section 4.3.2.

4.3.1 Moving Least Squares

For the resampling purposes considered here, a surface approximation technique for point-sampled geometry is required that ideally operates locally, efficiently, meshlessly and robustly in the presence of local non-uniformities of the point distribution. In the following section, I summarise Alexa et al. [4, 5] use of Levin's [92] extension of MLS function approximation to the approximation of $(m-1)$-manifold surfaces in $\mathbb{R}^m$, $m \geq 3$, for the design of a meshless surface approximation method for point clouds. The weighted least squares nature of this method accounts for its robustness when dealing with noisy input data [5]. In its localised, adaptive form, presented further below, it yields smooth surface approximations efficiently and robustly at the presence of local point distribution non-uniformities [124].

Moving Least Squares surface approximation

MLS surface approximation is based on a projection procedure. For a point near the surface, a support (tangent) plane to M near the point is computed first. The point is then projected onto a local polynomial approximation of M defined across the support plane. The original MLS procedure as introduced by Alexa et al. [4] for the approximation of a two-manifold surface in $\mathbb{R}^3$ represented by a point cloud P achieves this with the help of two weighted least squares computations. The first of these fits a local support plane H for $p_i \in P$ in the form of a weighted least squares best fitting plane by minimising the weighted distances of the input points from p_i, i.e.

$$\min n^T U n, \tag{4.1}$$

with normal $n \in \mathbb{R}^3$, $\|n\| = 1$, and $U = \{u_{vw}\} \in \mathbb{R}^{3x3}$ representing the weighted covariance matrix

$$u_{vw} = \sum_{j=1}^{\|P\|} \theta_j (p_{j_v} - p_{i_v})(p_{j_w} - p_{i_w}), \tag{4.2}$$

with

$$\theta_j = e^{-d(p_i,p_j)^2/\delta^2}, \tag{4.3}$$

where δ represents a global scale parameter. The value of δ will usually reflect any prior knowledge of the global sampling density such as the sampling resolution of the device used to acquire the point geometry; θ_j does not have to be a Gaussian but can be any function monotonically decreasing with distance. The normal n corresponding to H is determined

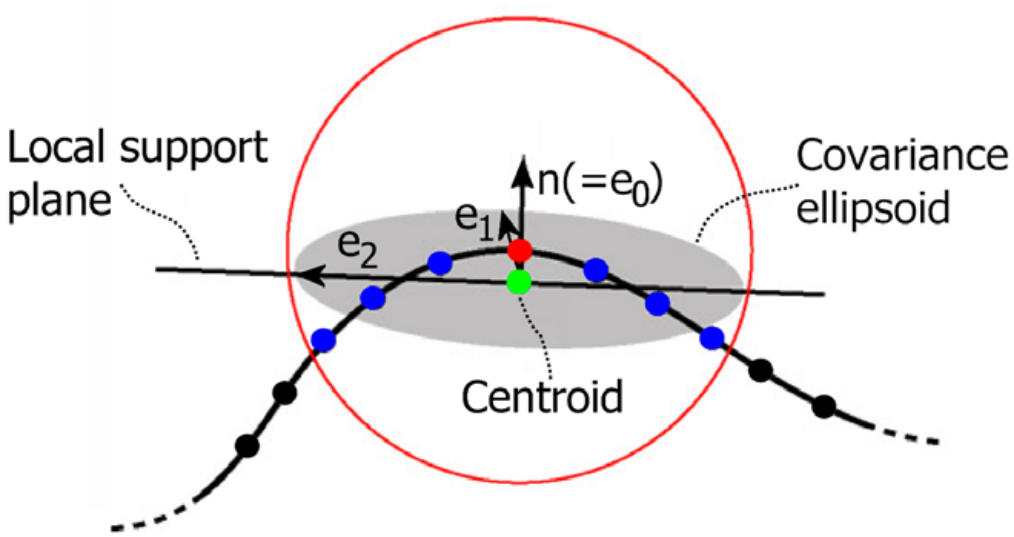

Figure 4.1: Eigenanalysis of the covariance matrix of the (blue) neighbours of a (red) point yields eigenvectors e_0, e_1 and e_2 defining an orthogonal frame centred at the (green) neighbourhood centroid and spanning a covariance ellipsoid. Under certain conditions, e_0 may be taken as normal estimate n defining this local (least squares best fitting) support plane. This view is in tangent direction.

using principal component analysis: Eigenanalysis of U yields its principal components in the form of orthogonal eigenvectors e_0, e_1, e_2 and corresponding real eigenvalues λ_0, λ_1, λ_2 spanning a covariance ellipsoid [72,124]. Provided that $\lambda_0, \lambda_1, \lambda_2$ are sufficiently dissimilar and $\lambda_0 \leq \lambda_1 \leq \lambda_2$, λ_0 describes the points' covariance along the local normal and e_0 may be chosen as local normal estimate [64] (Figure 4.1).[1] Subsequently, this estimate can be used to move H iteratively closer to the underlying surface. Alternatively, if the input point set is known to be relatively close to the underlying surface, the modification of H's position is not required [5].

Since the local support plane is therefore often close but not tangent to the underlying surface, the estimated normal generally does not coincide with the true surface normal. This does not represent a problem for the resampling purpose considered here since the normal estimate is used for the determination of the support plane and subsequent projection of sample points onto the MLS surface approximation. It is, however, discussed in more detail in the context of intrinsic meshless surface subdivision (Section 5.4.3) where the normal estimate needs to be used for orthogonal projection onto the underlying surface.

As part of the second weighted least squares computation, a neighbourhood of p_i is projected onto the support plane H. The neighbours' local 2D coordinates (x_i, y_i) in an orthonormal coordinate system on H and centred at the projection q_i of p_i are used to fit

[1] Section 4.5.1 discusses the case of ill-defined normal estimates due to the eigenvalue conditions being violated.

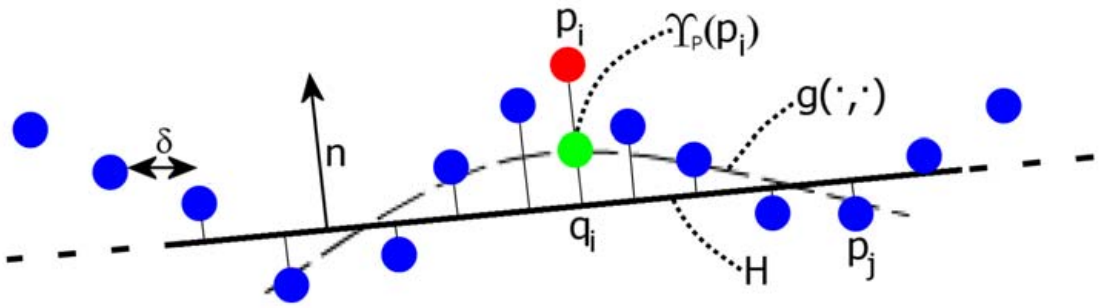

Figure 4.2: Moving Least Squares projection of a (red) point p_i; p_i is projected onto a bivariate polynomial $g(\cdot,\cdot)$ locally fit across support plane H to (blue) input points p_j weighted by a Gaussian weight function scaled by global sampling density estimate δ. The position of the projected point, i.e. $\Upsilon_P(p_i)$, is shown in green. This view is in tangent direction.

a bivariate polynomial $g(x_i, y_i)$ to the points in the neighbourhood. The weights used in the regression reflect the distance of the neighbours to q_i, i.e. $\theta_j = e^{-d(q_i,p_j)^2/\delta^2}$. The MLS projection of p_i given P then follows as (Figure 4.2)

$$\Upsilon_P(p_i) = q_i + g(0,0)n \tag{4.4}$$

With the help of the projection operator $\Upsilon_P(\cdot)$, the MLS surface approximation is defined implicitly as consisting of all points p_i which project onto themselves. Given a smooth weight function such as the Gaussian θ_j, Levin [93] conjectures the resulting MLS surface approximation to be globally smooth.

Although the MLS projection operator was discussed for a (possibly noisy) point $p_i \in P$, it applies equally well to any point $x \in \mathbb{R}^3$ which can be assumed to be located near the surface represented by P. In this case and given access to the neighbours of x in P, the above computation of the MLS projection operator carries over analogously.

Section 4.5.1 provides details of my implementation of the MLS algorithm.

MLS surface approximation implicitly assumes the absence of any non-uniformities in the distribution of P. This assumption expresses itself in the use of a global scale parameter δ. In the case of distribution non-uniformities, however, the continuing use of a global feature size δ will result in poorly fitting regressions. Note in this context that laser range scanning artefacts in particular can result in point cloud surface representations featuring hundreds of holes [65]. In these cases, a localised version of the MLS projection procedure which allows for variations in point density and thus an adaptive scale parameter for resampling and hole-filling purposes should therefore be used instead. Such a scale parameter is presented in the following section.

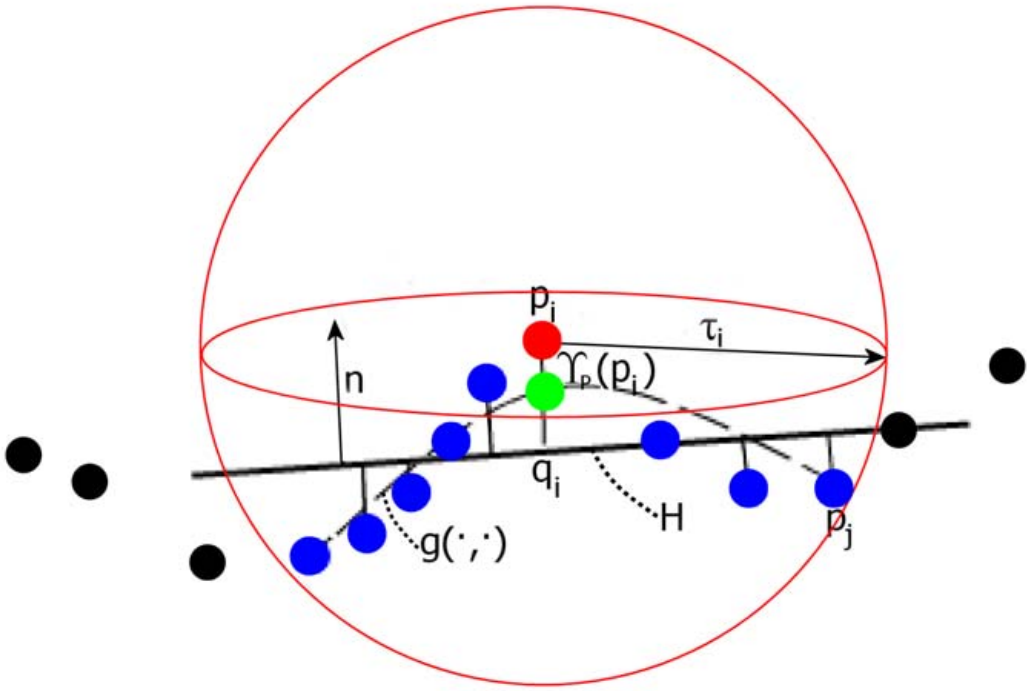

Figure 4.3: Adaptive Moving Least Squares projection of a (red) point p_i; p_i is projected onto a bivariate polynomial $g(\cdot,\cdot)$ locally fitted across support plane H to a (blue) neighbourhood of (otherwise black) non-uniformly distributed input points p_j; the neighbours are weighted by a quadratic B-spline centred at p_i and having spherically compact support of τ_i, the radius of the (red) sphere enclosing p_i's neighbourhood. The position of the projected point, i.e. $\Upsilon_P(p_i)$, is shown in green. This view is in tangent direction.

Localised weighting for adaptive MLS approximation

As typical for weighted least squares computations, in regions of locally small point density, a small value for the global scale parameter δ results in erroneous regressions due to insufficient input data. Similarly, when selecting a large δ-value when performing MLS computations in a region of high point density, geometric detail is smoothed out by the Gaussian kernel θ_j. Thus, the MLS approximation should be localised by varying the value of the scale parameter with a measure of local sampling density. This way high-detail information is preserved in areas of high point density and numerical stability is ensured in areas of low point density.

To allow for local changes in point density, following Ohtake et al. [119], the weight function of a neighbour p_j, θ_j, is determined by the following quadratic B-spline, $B(t)$, centred at the point p_i to be projected,

$$\theta_j = B\left(\frac{3d(p_i, p_j)}{2\tau_i}\right), \tag{4.5}$$

with $\tau_i = \max_{p_j \in eNN_{p_i}} d(p_i, p_j)$ representing the radius of the sphere centred at p_i and enclosing its enhanced k nearest neighbourhood, eNN_{p_i} (Figure 4.3); τ_i thus varies with local point density and defines the spherically compact support of the B-spline. The experimental results of Section 4.4 show this to be an useful choice of weight function in the context of this application.

Non-uniformity of the point set distribution not only affects the weighted least squares regressions of MLS but also requires an adaptively fitted offset band $\Omega_P^{r_i}$ and raises the question of the filling of illegitimate holes. My approach towards dealing with these issues is discussed in the following section.

4.3.2 Determination of offset ball radii and hole-filling

Adaptive offset band computation has been discussed in the context of my implementation of Fast Marching for surfaces in point cloud form (Section 2.5.2). This process is performed during the simplification algorithm's one-off pre-processing step. It determines local proximity information in the form of the enhanced k nearest neighbourhood of a point p, eNN_p, and then determines the Euclidean distance between p and its neighbour $q \in eNN_p$ farthest away from p. Provided $d(p, q)$ is larger than any radius currently associated with p and q, $d(p, q)$ is the new radius of both the offset balls centred at p and q. The grid vertices enclosed by these balls are included in the offset band. This can be considered as a (pessimistic) estimate of the local sampling density and is reminiscent of methods for the determination of the extent of surface elements ("surfels") for hole-free surface splatting, see, e.g. Wu and Kobbelt [174].

This first approach does not pay any attention to the risk of connecting disjoint sheets of the underlying surface. It may further bridge legitimate holes on manifolds with boundary. To avoid these problems, a maximum permissible radius is requested from the user. If, due to the size of a hole, the algorithm cannot determine an enhanced k nearest neighbourhood for this global radius, p may represent a border point. Alternatively, an undersampled region has been encountered which is too large to be bridged by eNN_p subject to the global radius maximum. The user is therefore asked to label p as a border point or to indicate that an illegitimate hole has been encountered. In the latter case, the algorithm increases the radius maximum locally until a valid enhanced k nearest neighbourhood has been found.

Following this pre-processing step, the offset band stretches across undersampled regions. When processing such regions, there do not exist any input points near samples located

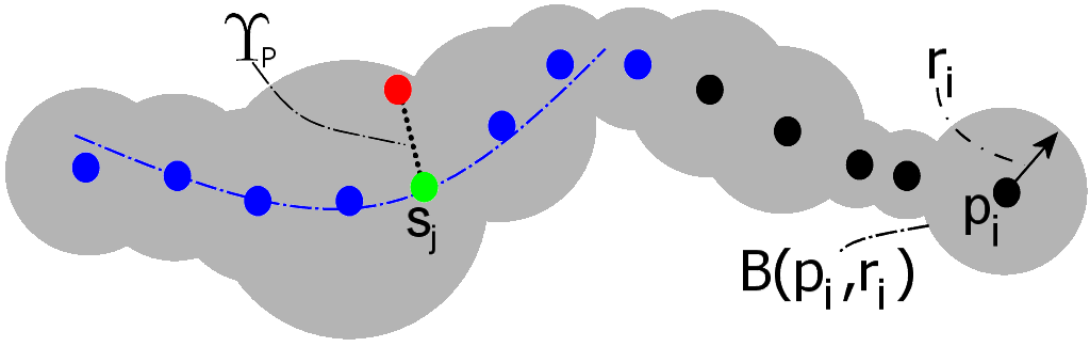

Figure 4.4: The offset radii r_i are adapted to changes in sample density. The r_i-values follow from the proximity information given by the point's enhanced k nearest neighbourhood. Simple holes are filled by projecting sample s_j from its (red) grid location in the bridging offset band onto the approximated surface (blue curve) generated by adaptive MLS (Figure 4.3). This view is in tangent direction.

in parts of the offset band which bridge illegitimate holes. These holes are filled by the adaptive MLS-based point cloud resampling technique. During resampling, a bivariate polynomial is fitted across the hole with the help of the surrounding input points in the sample's enhanced k nearest neighbourhood. The samples are then projected onto this polynomial (Figures 4.3, 4.4).

The following section gives a number of application examples for the intrinsic meshless sub- and resampling algorithms.

4.4 Experimental results

The intrinsic meshless simplification algorithms introduced in the previous sections are applied to a number of massive data sets and results for both uniform and feature-sensitive sub- and resampling are presented. The hole-filling capability of the resampling algorithm is also illustrated and the algorithms' inherent support of level-of-detail generation is highlighted. The section concludes with an analysis of the *a posteriori* approximation error introduced and a set of performance results.[2]

Figure 4.9 shows the distributions and surface reconstructions[3] of the Michelangelo Day and Michelangelo Dawn point sets both intrinsically uniformly subsampled to 1.0% of their size. The simplified point sets are irregularly uniformly distributed. This results in

[2]All processing was performed on a 1.0GHz AMD machine with 1GB of memory running under MS Windows 2000.

[3]This and all subsequent meshing of the sample points was performed with the help of PointsToPolys [130].

a cluster- and hole-free covering so that high quality further processing such as surface reconstruction and rendering is supported.

This favourable distribution property and the algorithm's coarse-to-fine progressive nature can be exploited for the generation of continuous level-of-details. This application is illustrated in Figure 4.10 for the Michelangelo Youthful point set. Similar to the mesh-based processing case (Section 3.7.3), this feature can be utilised for, amongst other things, the progressive transmission of 3D content in point cloud form.

To give an application example for the intrinsic meshless non-uniform simplification of a surface, I estimate local changes in mean curvature by the local surface variation measure of Pauly et al. [124] to drive the curvature-sensitive subsampling of the Venus point set. The surface variation measure is here based on the principal component analysis of the covariance matrix (4.6) below of an enhanced k nearest neighbourhood of the point p under consideration. As briefly discussed in Section 4.3.1, given that the eigenvalues $\lambda_0, \lambda_1, \lambda_2$ of this matrix are sufficiently dissimilar and $\lambda_0 \leq \lambda_1 \leq \lambda_2$, λ_0 describes the deviation of the neighbours from the eigenvector corresponding to λ_0, i.e. from the estimated normal vector at p. Pauly et al. [124] define surface variation as the ratio of this deviation to the total deviation, i.e. as $\lambda_0/(\lambda_0 + \lambda_1 + \lambda_2)$, where λ_1 and λ_2 describe the variation along the other two eigenvectors of the covariance matrix, i.e. the variation in the local support plane (Figure 4.1). Note in this context that since the eigenvalues vary with the size of the point neighbourhood, this ratio does not represent a particularly robust approximation of mean curvature. Nevertheless, as illustrated in Figure 4.11, by varying propagation speeds with this curvature estimate, non-uniform subsampling more strongly concentrates the samples in regions of strong curvature. Provided the speed remains strictly positive throughout, non-uniform sampling may be driven by any other or the combination of additional adaptivity measures. Excessive non-uniformity of the resulting point set can be avoided in this context by enforcing a globally defined refinement condition ρ.

The non-uniform sampling may also be driven by a more direct geometric accuracy measure to improve the fidelity of the simplification. To illustrate this, Alexa et al. [5] MLS-based redundancy measure is computed for each point of the Venus point cloud and then imported as an importance map by the implementation of the weight function module. More specifically, the importance of $p_i \in P$ is measured by computing $\Upsilon_{P\setminus\{p_i\}}(p_i)$, i.e. the MLS projection of p_i onto the MLS surface approximation computed from $P \setminus \{p_i\}$. The distance of p_i from this projection determines the relevance of p_i for the definition of the shape represented by P. The larger this distance, the higher the relevance of p_i. Figure 4.12 presents the importance map computed this way for the Venus point set and the resulting non-uniform subsamplings. Figure 4.13 gives the results of the analysis of

Figure 4.5: Optical acquisition of a Buddha sculpture using laser range scanning. The object was rotated by 60 degrees after each range shot taken with a Minolta VIVID-900 laser scanner. The scanner's precision of 0.02mm resulted in a representation of the sculpture's surface consisting of 152154 points. This representation was affected by artefacts due to shortcomings inherent to optical acquisition and suboptimal acquisition conditions.

the *a posteriori* approximation error introduced by the sampling using, for simplicity, the Metro tool [32] for numerical and the MeshDev tool [137] for visual evaluation. To be able to apply these mesh-based tools, the simplified point clouds were meshed using PointsToPolys [130]. For comparative purposes, the results produced by Garland and Heckbert's [53] quadric error mesh simplification algorithm QSlim 2.0 are also shown. The simplified meshes produced by QSlim contain slightly more vertices due to the difficulty of controlling the vertex count in the simplified model. More importantly, since the samplings produced by my subsampling algorithm are fitted to a smooth MLS approximation, they do not fit the piecewise linear nature of the original, non-simplified Venus mesh referred to by the evaluation tools as well as the simplified mesh produced by QSlim. The results produced by QSlim are, however, less superior than could be expected given the different continuity classes of the underlying surface approximations.

To give an example for the hole-filling capability of the intrinsic resampling algorithm, the surface of the Buddha sculpture of Figure 4.5 was optically acquired using a state-of-the-art Minolta VIVID-900 laser range scanner. The various range scans of the sculpture were obtained with the help of a simple rotary table turned by 60 degrees after each shot. The top and bottom of the sculpture were scanned separately. As a result, a number of surface concavities were missed by the laser. Together with the suboptimal lighting

	Input size	Output size	Execution time (secs.)
Venus	134345	1343	3.37
Isis	187644	1876	7.04
Michelangelo Youthful	1728305	17283	22.50
Michelangelo Day	3158672	31587	40.64
Michelangelo David	6924951	69250	332.65
St. Matthew	11879867	118799	688.28
Lucy	14027872	140279	997.24

Table 4.1: Execution times for the 99% simplification of various input point clouds using uniform intrinsic meshless surface subsampling.

conditions during the scanning process, a number of artefacts were thus generated in the form of measurement outliers and undersampled regions. These artefacts are also typical of more sophisticated acquisition procedures than the one employed here [141]. Whilst the postprocessing software accompanying the VIVID-900 dealt successfully with a number of the artefacts, the triangular mesh returned by the software nevertheless featured a genus significantly larger than that of the original object. These illegitimate holes proved simple enough to be filled using automatic adaptive offset band generation followed by uniform resampling of the mesh vertices stripped of their connectivity. As a result and as illustrated in Figure 4.14, following the uniform resampling of the acquired point set, meaningful further processing such as surface reconstruction becomes possible.

Finally, as documented in Table 4.1 and Figure 4.6 respectively, the algorithm's execution time is only moderately affected by substantial increases in input or output point cloud size. Apart from the case of the Lucy point set and due to the consideration of offset band grid points only, only a fraction of the available memory was used by the algorithm at any one point. This is in contrast to grid-based techniques such as that of Guy and Medioni [66] which discretise a point set's bounding volume at the expense of prohibitively large memory demands for relatively small point sets by today's standards.

4.5 Implementation details

Section 4.5.1 provides details on my implementation of MLS surface approximation. My implementation of enhanced k nearest neighbourhood computation is discussed in Section 4.5.2.

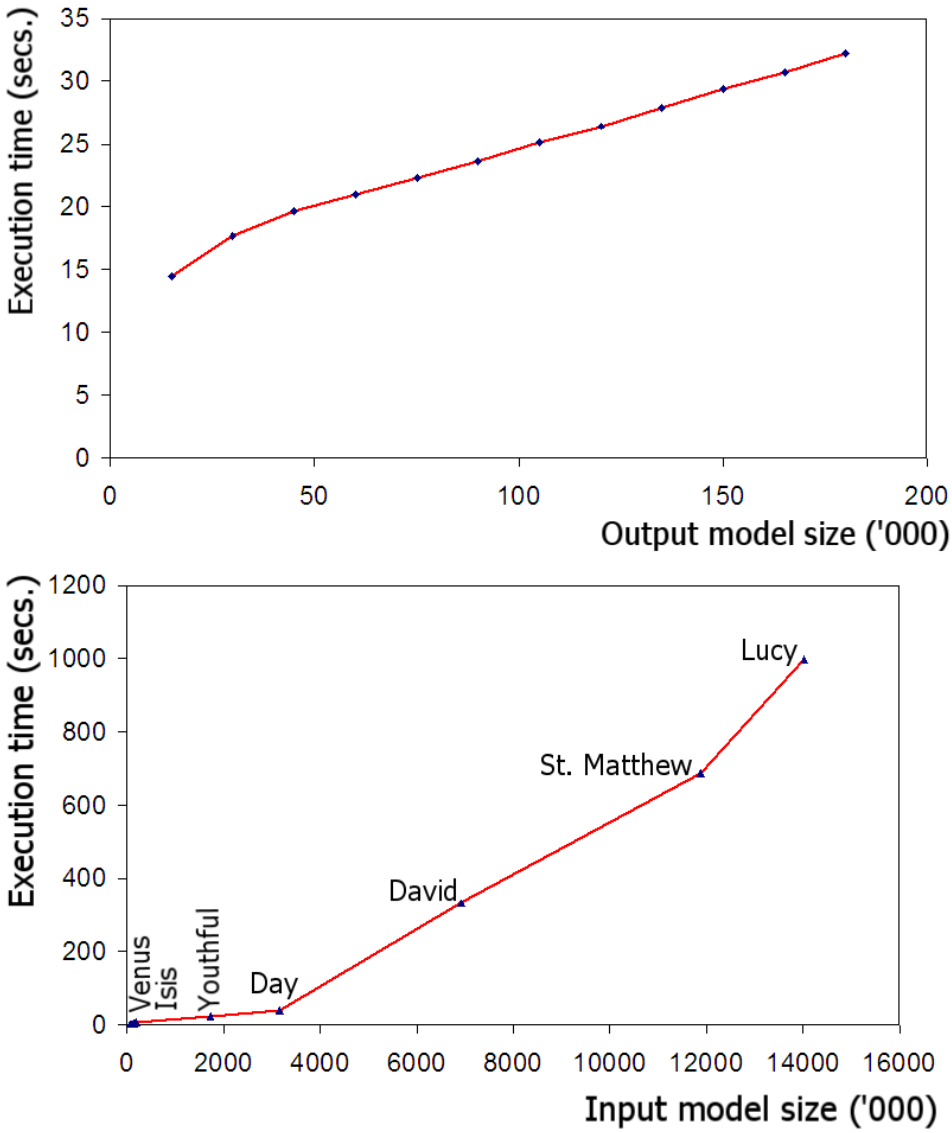

Figure 4.6: Uniform simplification execution times as function of output (*top*) and input point set size (*bottom*). For the former, the Isis point cloud was uniformly subsampled to different output sizes ranging from 15k to 180k samples. The results are representative for the execution efficiency of uniform subsampling when dealing with any other of the test point sets. For the latter, each input model was uniformly subsampled to 1% of its size (Table 4.1).

4.5.1 Moving Least Squares

The MLS weighted least squares regressions of Section 4.3.1 have been implemented with the help of the newmat11 matrix library [117]. My code first sets up the weighted covariance matrix U (4.2). Using the eigenanalysis function of newmat11, it then estimates the (normalised) normal n of p_i's support plane H in the form of the eigenvector of U corresponding to its smallest eigenvalue. The input point sets considered in the previous

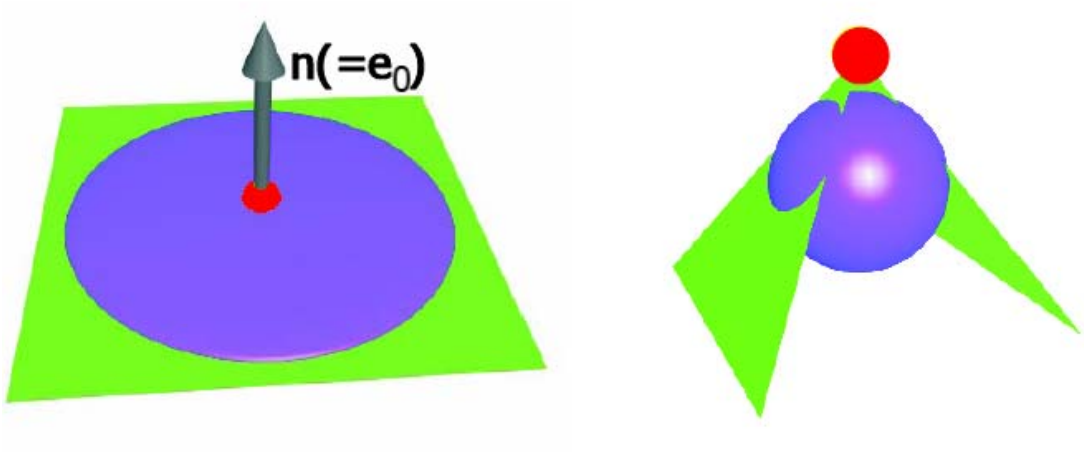

Figure 4.7: In the case of a "normal" (red) surface point, the normal estimate for the (green) surface obtained from principal component analysis is well-defined (*left*). When trying to estimate the normal at a corner point, the covariance ellipsoid has no preferred direction and no eigenvector can be selected as a normal estimate (*right*). This figure is adapted from [64].

section were found to be close enough to the underlying surface that no further non-linear optimisation of the position of H was required.

To set up the orthonormal coordinate system centred at the projection q_i of p_i, the vector from the first neighbour's projection onto H to q_i is taken as the x-axis. The y-axis then immediately follows as the cross product of the x-axis and the normal vector n. The projected neighbours' coordinates in this local coordinate system are used to fill up the coefficient matrix of a bivariate cubic polynomial to be fitted by weighted least squares to the neighbourhood across H. The corresponding normal equations are solved directly with the help of newmat11's matrix algebra operators. The MLS projection follows from the value of this polynomial at the origin.

At corner points, the covariance ellipsoid defined by the eigenvectors and eigenvalues of U (4.2) will feature no preferred direction which expresses itself in eigenvalues of similar values [64]. In this case, none of the eigenvectors can be chosen as normal direction and n is undefined (Figure 4.7). MLS surface approximation is, however, derived under a smoothness assumption as regards the underlying surface [93]. If this assumption does not hold, the MLS projection is ill-defined and a singularity in the surface approximation results. If these singularities are deemed undesirable, the implementation offers an interactive mode which asks the user for the normal direction whenever principal component analysis fails to yield a well-defined normal vector.

Along creases/edges, the covariance ellipsoid is stretched in the direction of the crease and it is $\lambda_0 \approx \lambda_1$ and $\lambda_0 + \lambda_1 \approx \lambda_2$ [64]. The average of the eigenvectors corresponding to

λ_0 and λ_1 may then be used as a rough normal estimate. This automatic choice can be modified interactively.

4.5.2 Enhanced k nearest neighbourhood

The computation of an enhanced k nearest neighbourhood eNN_{p_i} is performed by first growing the conventional k nearest neighbourhood NN_{p_i} until it is comprised of a preset minimum number of neighbours k. My informal experimental results indicate a choice of $8 \leq k \leq 18$ for the neighbourhood size to be well-suited for most cases. The local normal vector is then estimated with the help of newmat11 [117] by first computing the positive semi-definite weighted covariance matrix $C = \{c_{vw}\} \in \mathbb{R}^{3x3}$, of the points $q_j \in eNN_{p_i}$ around their centroid $c_{eNN_{p_i}}$,

$$c_{vw} = \sum_{j=1}^{k} \theta_j (q_{j_v} - c_{eNN_{p_i}})(q_{j_w} - c_{eNN_{p_i}}), \tag{4.6}$$

with $\theta_j = B\left(\frac{3d(c_{eNN_{p_i}}, q_j)}{2\tau_c}\right)$, where $\tau_c = \max_{q_j \in eNN_{p_i}} d(c_{eNN_{p_i}}, q_j)$ and $c_{eNN_{p_i}} = 1/k \sum_{j=1}^{k} q_j$, $k = \|eNN_{p_i}\|$. This definition of $c_{eNN_{p_i}}$ needs to be modified when dealing with relatively strongly non-uniformly distributed point sets. In this case, $c_{eNN_{p_i}}$ may be defined as the weighted centroid minimising a weighted sum of squares of Euclidean distances [24],

$$f(c_{eNN_{p_i}}) = 1/2 \sum_{i=1}^{k} w_i d(c_{eNN_{p_i}}, q_i)^2, \tag{4.7}$$

where $w_i = \theta_i / \sum_{j=1}^{k} \theta_j$. Differentiation of (4.7) then yields $c_{eNN_{p_i}} = \sum_{i=1}^{k} w_i q_i$ (Section 5.2.2).

Similar to the MLS technique, provided the eigenvalues of C are both sufficiently dissimilar and $\lambda_0 \leq \lambda_1 \leq \lambda_2$ holds, the eigenvector e_0 of smallest eigenvalue λ_0 is taken as the estimate of the local normal vector; otherwise, the estimate is unreliable and the growing of eNN_{p_i} resumes.

The normal vector estimate is subsequently used to project the q_j into the local support plane. The algorithm determines the coordinates of the q_j in an orthonormal coordinate system across the plane and centred at p_i. These coordinates are transformed into polar coordinates and it is then controlled for the difference in polar angle between successive neighbours in counter-clockwise order around origin p_i. If none of the angles between successive neighbours is found to be larger than the threshold, a valid eNN_{p_i} has been

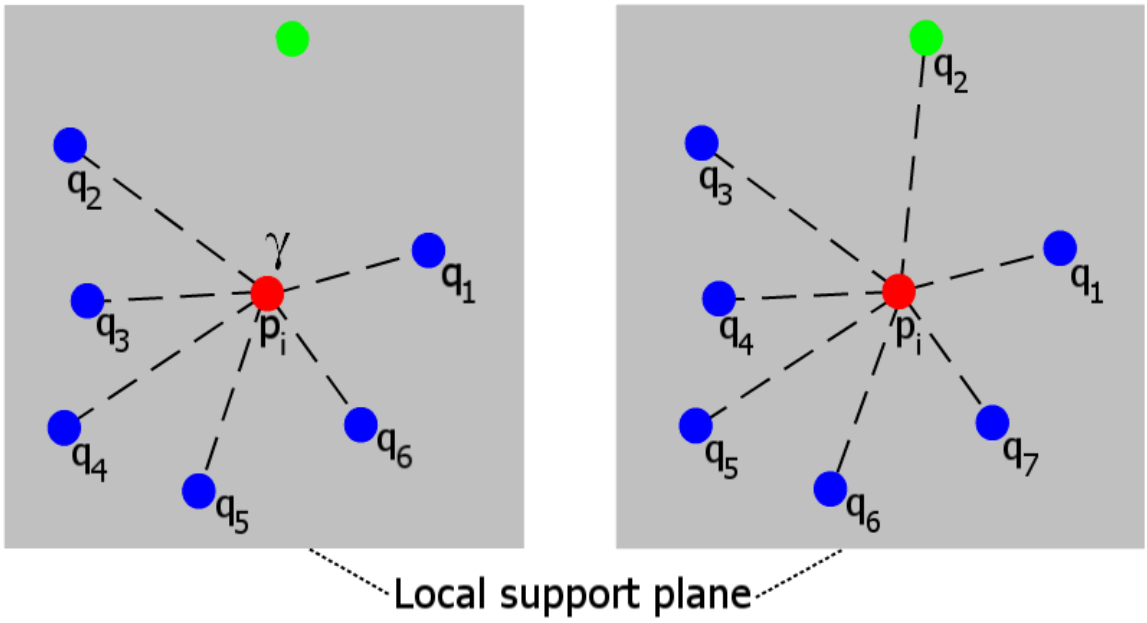

Figure 4.8: The enhanced k nearest neighbourhood of a (red) point p_i is computed by projecting its conventional k nearest neighbours into a (grey) local support plane defined by p_i and its normal estimate. If the angle between successive neighbours exceeds a maximum of γ (*left*), additional neighbours are considered until either a suitable (green) neighbour has been found (*right*) or it has been established interactively that p_i represents a border point. This view is in normal direction.

found; otherwise, the algorithm continues growing the neighbourhood until neighbours have been found which do not violate the angular threshold (Figure 4.8). If this does not prove possible until the preset maximum of conventional k nearest neighbours has been considered, p_i may represent a border point. As in the case of adaptive offset band fitting, this issue and the issue of ill-defined normal estimates are resolved interactively with the help of the user.

This process is prone to the connection of disjoint surface sheets and the bridging of legitimate holes. Note, however, that, apart from the computation of the offset band $\Omega_P^{r_i}$ itself, the enhanced k nearest neighbourhoods are computed inside $\Omega_P^{r_i}$, i.e. intrinsically thereby yielding enhanced k nearest neighbourhoods in a geodesic sense. Thus, provided $\Omega_P^{r_i}$ has been fitted successfully, the above problems will generally be avoided during the computation of enhanced k nearest neighbourhoods. This computation may, however, still result in poorly defined neighbourhoods in cases of extreme geometry.

	Density guarantee	Guaranteed approx. error	Flexible adaptive simplif.	Sub- and resampling support	Computat. efficient	Memory efficient	Progressive
Alexa et al. [4]	No	Yes	No	No	No	Yes	No
Dey et al. [41]	Yes	Yes	No	No	No	No	No
Linsen [97]	No	No	Yes	No	Yes	Yes	No
Moenning and Dodgson [112]	Yes	No	Yes	No	Yes	Yes	Yes
Pauly and Gross [123]	Yes	Yes	No	Yes	No	Yes	No
Pauly et al. [124] -							
Iterative simplification	No	Yes	No	Yes	Moderately	Yes	No
Incremental clustering	No	No	No	No	Yes	Yes	No
Hierarchical clustering	No	No	No	No	Yes	Yes	No
Particle simulation	No	No	Yes	Yes	No	Moderately	Yes
Intrinsic point cloud simplif.	Yes	No	Yes	Yes	Yes	Yes	Yes

Table 4.2: Comparative evaluation of the intrinsic point cloud simplification algorithm presented in this chapter (*bottom row*) and existing alternative methods.

4.6 Summary and discussion

In this chapter, I presented intrinsic meshless sub- and resampling algorithms for surfaces represented by point clouds. The algorithms support the uniform and adaptive simplification of uniformly distributed point sets and point clouds featuring non-excessive non-uniformity and/or illegitimate holes of simple complexity in the form of undersampled regions. Adaptivity is supported in the form of any combination of (positive) point weights either computed on-the-fly or imported in the form of pre-computed importance maps. The algorithms' coarse-to-fine progressive nature inherently supports the generation of continuous level-of-detail representations and progressive transmission of 3D content. The sub- and resampling methods inherit the sampling distribution and guaranteed density properties of their parent template. They have been shown experimentally to support the efficient simplification of large point clouds in-core. As summarised in Table 4.2, the technique put forward in this chapter represents the first simplification algorithm combining simple control of guaranteed density with this set of features.

Although my hole-filling approach supports the processing of the type of moderately non-uniformly distributed point cloud data frequently encountered in practice, it does not allow for the processing of complex or sizeable holes since the adaptive offset band may not enclose the whole of the surface in such cases. This is due to the fact that the enhanced k nearest neighbourhood concept used in this context to obtain local proximity information does not guarantee symmetric neighbour relations. That is, $p_i \in eNN_{p_j}$ does not imply $p_j \in eNN_{p_i}$, for $p_i \neq p_j$. As a result, p_i and p_j are likely to be assigned different

offset ball radii which may leave holes in the enclosure of the underlying surface. The corresponding grid vertices would be excluded from processing by the algorithms presented in this chapter so that no hole-filling would be performed. In this case, the technique presented here would benefit from a more sophisticated method for adaptive offset band fitting such as the use of minimum spanning trees or local principal component analysis as discussed in Mémoli and Sapiro [103,104]. However, for very complex or sizeable holes, the dedicated hole-filling techniques presented by Sharf et al. [149] and Weyrich et al. [169] are more suitable.

In the case of problems with respect to local normal estimation using principal component analysis or the determination of enhanced k nearest neighbourhoods, the algorithms lack automatism and rely on interactivity. For the majority of point sets processed here, this approach did not prove to be a problem. It is, however, error-prone and may place a relatively large burden on the user in the case of challenging geometry as experienced here in the case of the Michelangelo Day and Dawn point sets.

A simplified point cloud should generally be as close to the original point cloud as possible. In the case of scientific visualisation, for example, the preservation of appearance properties and a guaranteed *a priori* geometric approximation error are of foremost importance. In the case of real-time applications, applications generating level-of-detail hierarchies, point-based rendering and shape matching, performance, uniformity of the sampling distribution and a guaranteed, intuitively controllable sampling density respectively represent the more relevant criteria. The intrinsic point cloud sampling algorithms presented in this chapter are well-suited for these latter purposes. Mémoli and Sapiro [105,106], for example, exploit its guaranteed sampling density property for the generation of representative point cloud subsets in the context of point cloud-based manifold comparison.

Also note in this context that just as in the mesh simplification case, guaranteed approximation error control tends to come at the expense of a significant performance hit. In the case of Wu and Kobbelt's [174] splat subsampling algorithm with *a priori* geometric approximation error guarantee, for example, the most challenging application example, the Charlemagne point set of 598386 points, is only a fraction of the size of the Lucy point set (14027872 points) but the processing times of the two algorithms reported for these different point sets are similar, 935 [174] and 997 seconds respectively.[4]

Although the algorithms presented here do not guarantee an *a priori* geometric approximation error, it has been shown that approximation quality measures can be incorporated relatively easily in the form of an implementation model for the weight function module.

[4]The processing time of Wu and Kobbelt's [174] splat subsampling algorithm is reported for a Pentium 4 2.8GHz machine with 2GB of memory compared with my AMD 1.0GHz machine with 1GB of memory.

This has specifically been illustrated using Alexa et al. [5] redundancy measure based on MLS surface approximation. The resulting *a posteriori* approximation error has experimentally been found to be favourable indicating that the methods presented can be used for both efficient and accurate intrinsic meshless surface simplification.

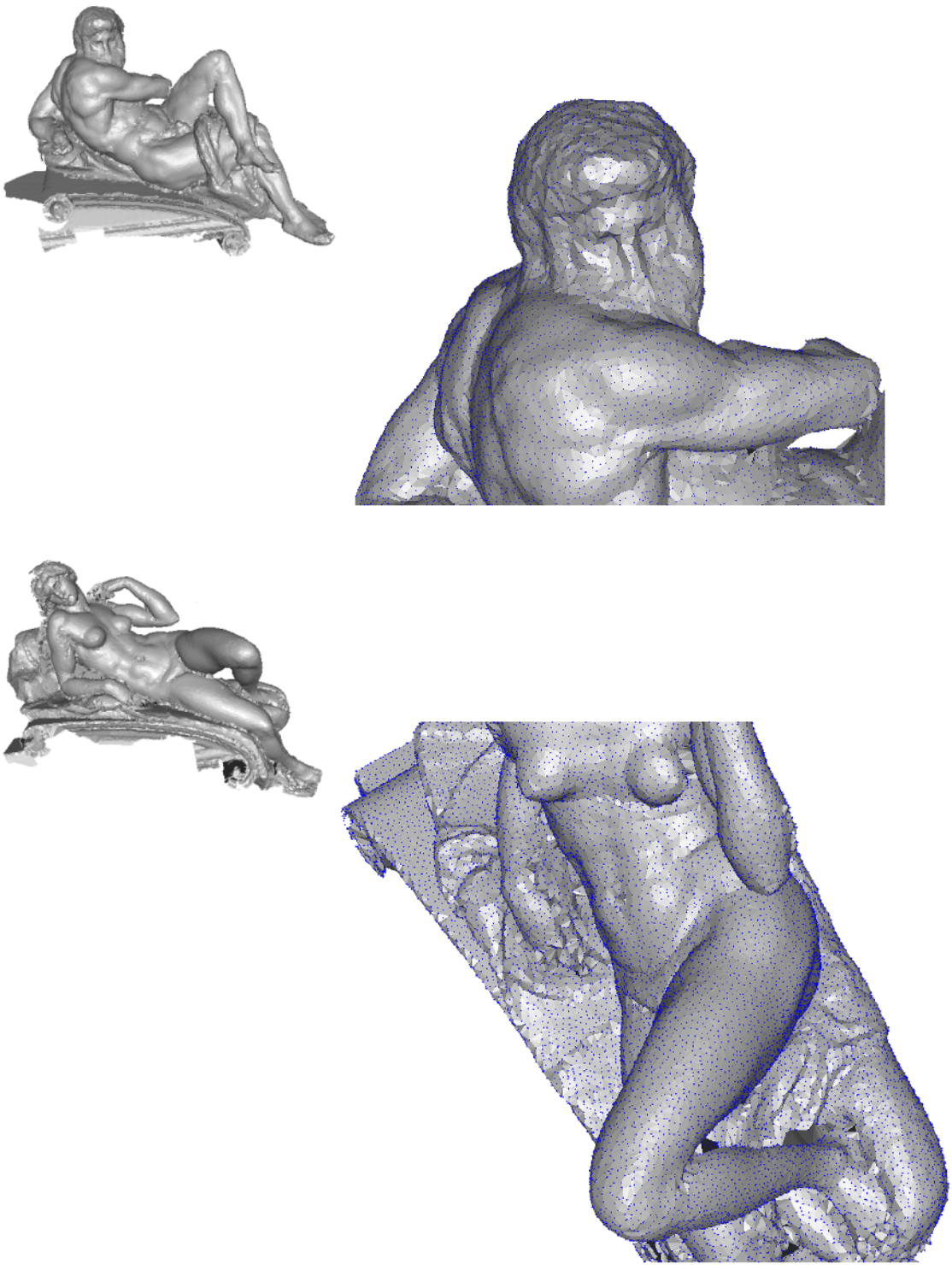

Figure 4.9: Examples for the quality of the generated sample distributions: The Michelangelo Day (*top*) and Dawn (*bottom*) data sets are uniformly subsampled to 1.0% of their original size of 3158672 and 3432236 points respectively.

Figure 4.10: Level-of-details of the Michelangelo Youthful data set (1728305 points) produced from point sets generated by progressive uniform resampling of the original point cloud to 0.25%, 0.5%, 1.0% and 5.0% of its size (*from left to right*).

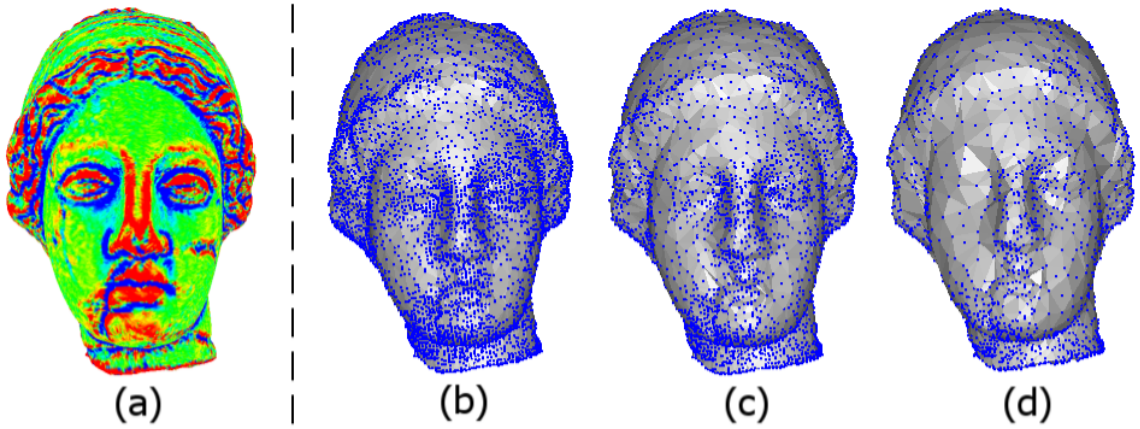

Figure 4.11: Adaptively subsampled Venus (134345 points) point sets driven by local surface variation estimates [124]. The model's mean curvature plot produced using Rapid-Form 2004 [132] is shown in (a) (blue indicates low, green indicates low-medium, yellow indicates high-medium and red indicates high curvature regions). The point sets correspond to 90.0% (b), 95.0% (c) and 97.5% (d) simplification.

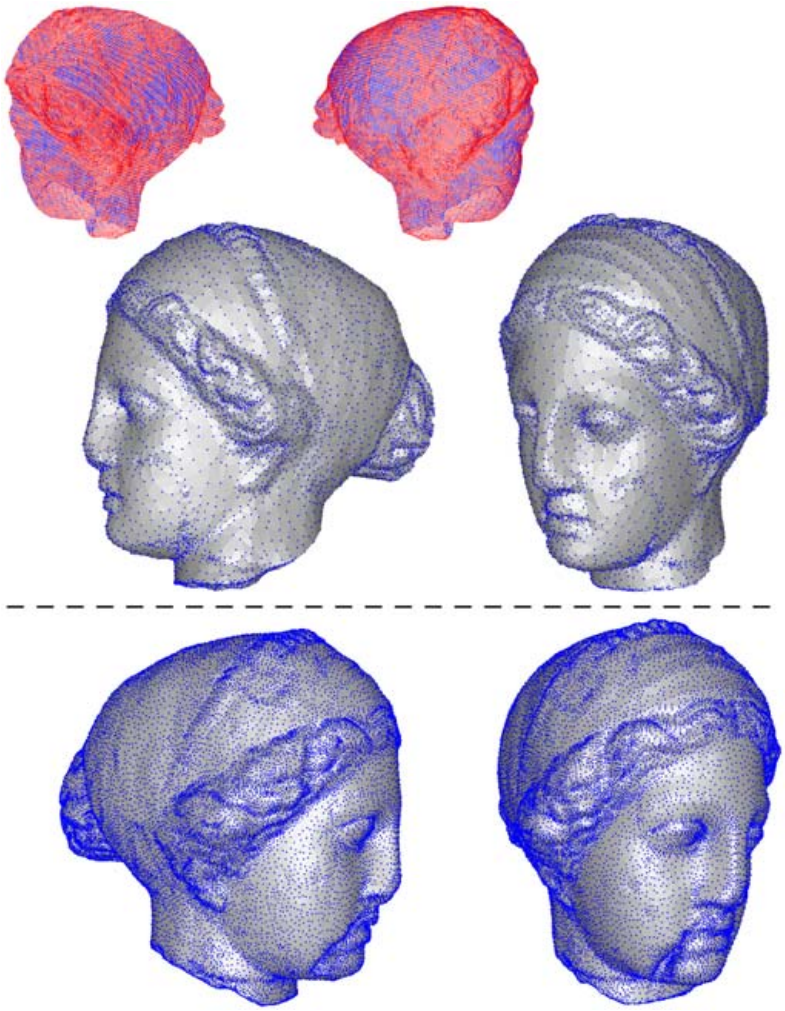

Figure 4.12: Adaptive subsampling of the Venus point cloud (134345 points) by a MLS-based importance measure to 7.44% (*top*) and 22.33% (*bottom*) of its input size. Blue indicates low and red indicates high importance in the two views of the importance map shown at the top. The points in the importance map renderings are not depth-buffered.

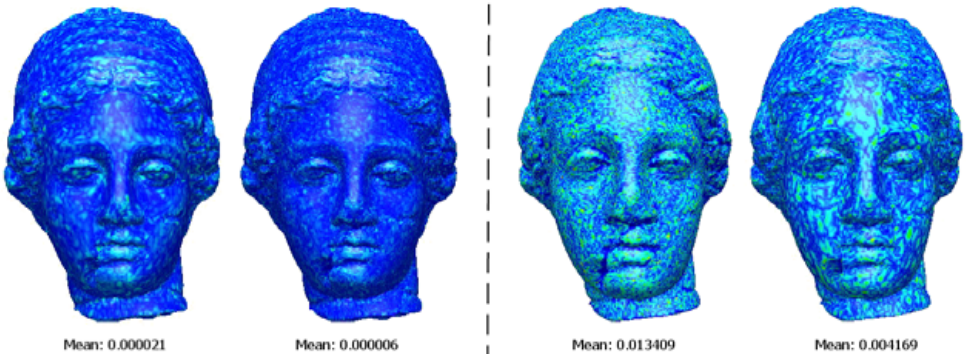

Figure 4.13: Visualisation using MeshDev [137] of the *a posteriori* approximation error introduced by QSlim [53] mesh simplification (*left*) and intrinsic point cloud subsampling followed by mesh reconstruction (*right*) respectively. Blue indicates low, green indicates medium and red indicates high error regions. The mean approximation error with regard to the mesh bounding box is given below each figure. The error represents the mean distance between the original and the simplified mesh as reported by Metro [32]. The simplified meshes consist of, from left to right, 10596 (7.89%), 30226 (22.50%), 10000 (7.44%) and 30000 (22.33%) vertices respectively.

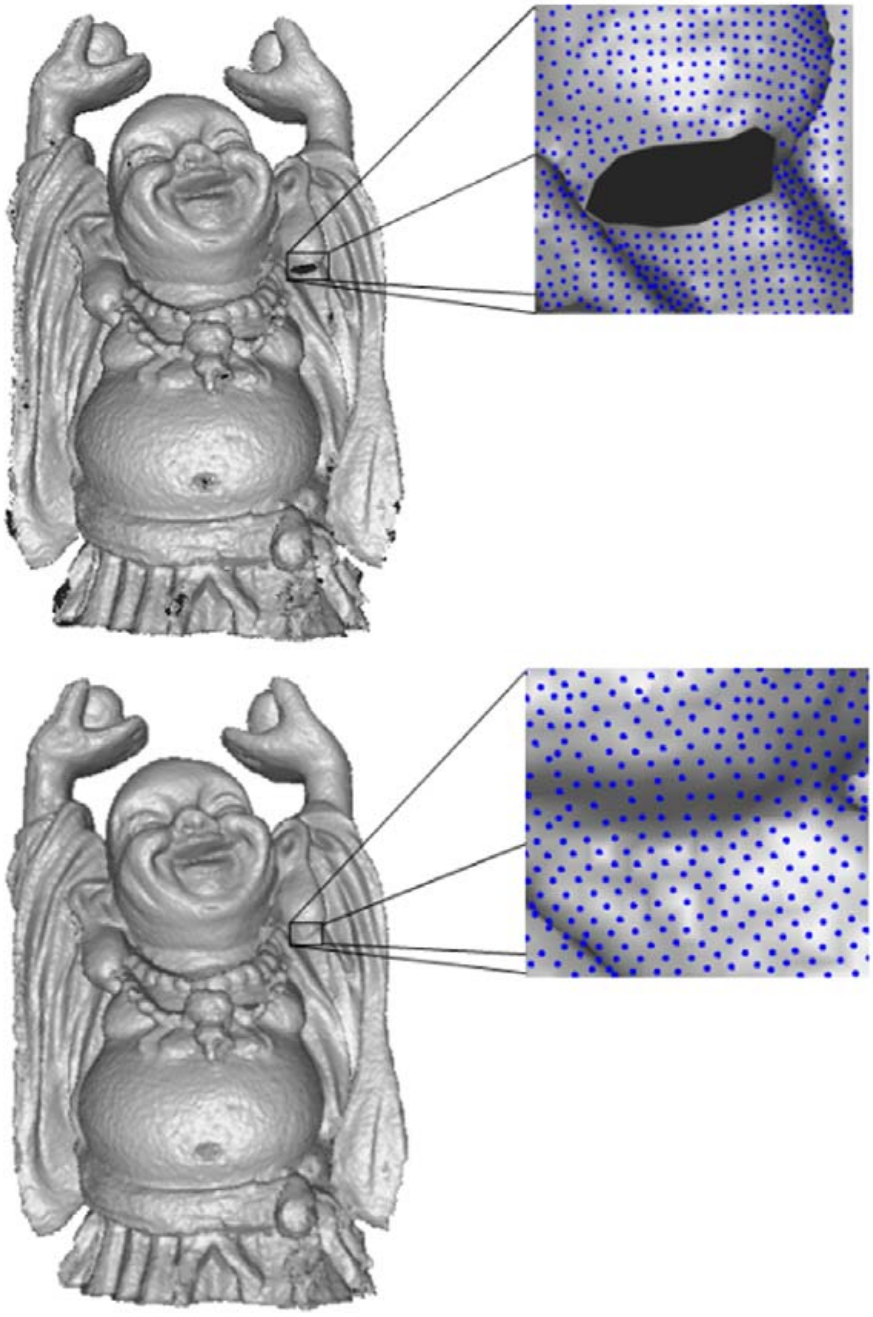

Figure 4.14: Simple hole-filling example for a Buddha point set (152154 points) acquired using a laser range scanner. Since subsampling of the data set would retain the holes (*top*), the point cloud is uniformly resampled instead (*bottom*) by computing an adaptive offset band and subsequent projection of sample points inside that band onto a local MLS surface approximation. The bottom view is zoomed-in more strongly to illustrate the effect.

Chapter 5

Intrinsic meshless surface subdivision

Mesh-based surface subdivision has developed into a powerful and widely-used tool for the free-form design, editing and representation of smooth surfaces. Mesh-based subdivision schemes recursively apply a local subdivision operator to a coarse base mesh thereby producing a sequence of refined meshes which quickly converges to a smooth limit surface. The advantages of mesh subdivision include guaranteed global surface smoothness whilst supporting local feature control, the ability to handle surfaces of arbitrary topology and being efficient and simple to apply once a base mesh is available. Typical application examples include the generation and rendering of smooth surfaces for CAD and animation. The link between mesh-based subdivision surface representations and wavelets is used for multiresolution surface editing and progressive transmission and compression [44, 177].

Unfortunately, when dealing with point-sampled geometry, mesh subdivision requires costly and generally non-geometric surface reconstruction [8–10, 15, 19, 38, 45], often followed by mesh simplification [58], parameterisation [51] and remeshing [6], all pre-processing steps to obtain a base mesh. During the surface reconstruction step, any measurement noise or data acquisition artefacts may translate into topological artefacts in the form of erroneous connectivity and genus [173]. This hinders subsequent mesh simplification and remeshing and thus mesh subdivision processing. In the case of extremely high-dimensional manifolds by samples [156], mesh-based processing fails at the surface reconstruction step and any surface processing needs to work directly with the point cloud. The most widely-adopted mesh subdivision schemes such as Catmull-Clark [26] for quadrilateral and Loop [98] subdivision for triangular meshes are restricted in applicability to one type of mesh element only, quadrilaterals or triangles. The resulting subdivision surfaces represent extensions of uniform box-spline surfaces and cannot easily be made geometry-sensitive, i.e. non-linear.

This chapter advocates the use of intrinsic meshless, or point cloud, surface subdivision. It is proposed to avoid the consideration of mesh connectivity graphs and instead to work with the point-sampled geometry directly using intrinsic subdivision rules. We introduce an intrinsic meshless surface subdivision scheme using weighted geodesic centroids of intrinsic natural neighbourhoods (Section 2.4.3) and put forward a new method for the computation of these geodesic means on manifolds, which by itself is of interest in other areas such as intrinsic statistical shape analysis [50] and variational theory [75, 81].

Some of the related geometric operations may be approximated in an Euclidean context when working with large sampling densities, regular meshes and very local subdivision rules. Working with the point-sampled geometry directly and intrinsically, however, avoids the need for non-geometric pre-processing steps and special rules dealing with irregular mesh connectivity. Intrinsic meshless surface subdivision tends to be more generally applicable in that geodesic averaging rules can be non-local and do not vary with the particular type of data (mesh) representation used. Finally, unlike the non-geometric nature of Euclidean-based (pre-)processing, intrinsic meshless surface subdivision inherently captures the non-linear intrinsic structure of the object geometry and the principle applies equally well to three and higher-dimensional surfaces. Although it is possible to obtain a geodesic mesh representation of a point-sampled surface [127, 152], to continue performing subdivision truly intrinsically, this mesh would have to be modified repeatedly during each iteration to re-enforce its intrinsic nature and more than one mesh would be required to be able to compute correct geodesic distances.

The algorithm operates intrinsically throughout without the need for prior surface reconstruction with the help of the intrinsic distance mapping algorithm of Mémoli and Sapiro [104] (Section 2.3.3). Following an overview of related work in Section 5.1, the intrinsic meshless surface subdivision algorithm is presented in Section 5.2. Section 5.3 gives experimental results and Section 5.4 comments on implementation aspects. Section 5.5 concludes this chapter with a summary and a discussion of the results.

This chapter is based on Moenning et al. [114, 115].

5.1 Related work

I start with a summary of the ideas underpinning mesh-based subdivision of surfaces in $\mathbb{R}^3$. The overview is motivational in nature, for a thorough formal treatment of mesh subdivision, see Dyn and Levin [44]. The section is concluded with remarks on recent progress in meshless, point-based surface processing related to this work.

Following the notation of Dyn and Levin [44], mesh-based surface subdivision schemes consist of a subdivision operator $\mathcal{S}$ recursively applied to (control) meshes $N_l = (V^l, E^l, F^l)$ of arbitrary topology, with $l \in \mathbb{Z}_0$ denoting the subdivision level, V representing a set of control vertices in $\mathbb{R}^3$ and E and F describing the topological relations in the form of edges and faces respectively. The iterative application of this scheme generates a sequence $N_{l+1} = \mathcal{S}N_l$. More specifically, starting with a coarse base mesh N_0, at each iteration, new control vertices are inserted and connected according to the scheme's refinement rule and re-positioned following the operator's geometric averaging rule. Both the refinement and the geometric averaging rule give the position of control vertices in N_{l+1} in the form of weighted averages of topologically neighbouring vertices in N_l. The careful choice of these weights in relation to the control vertex valency, i.e. the number of edges emanating from the vertex, guarantees the convergence of the scheme, in each component and in the uniform norm, to a limit surface of provable continuity.

Not every existing mesh subdivision operator allows for this simple distinction between a topological refinement and a geometric averaging rule. However, those that do allow for this kind of distinction, include the most widely-used schemes. For example, the Loop [98] subdivision scheme for triangular meshes may be cast in this form. In the case of this scheme, the refinement rule adds points related to each edge in the triangulation using face splitting. The averaging rule of points, which depends on the vertices and edges of the non-refined triangulation, then yields the final positions of the vertices in the refined triangulation (Figure 5.1). Figure 5.2 gives an example of Loop subdivision.

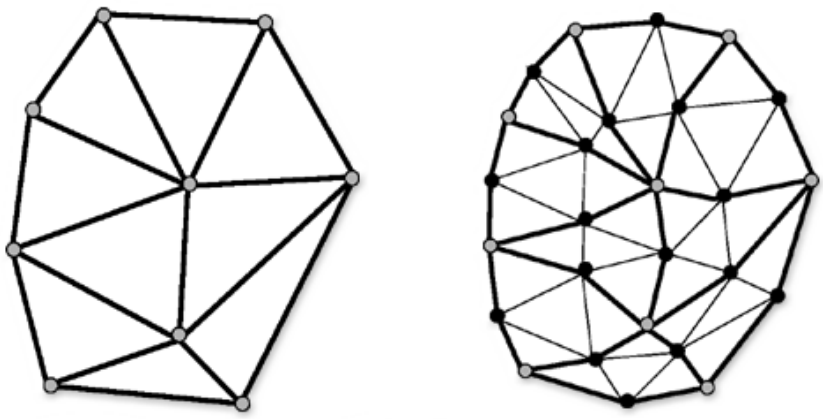

Figure 5.1: Loop subdivision scheme for triangular control meshes: The refinement rule adds (black) points related to each edge in the control mesh (shown on the left) using face splitting. The averaging rule of points, which depends on the vertices and edges of the control mesh, then gives the final positions of both the original (grey) vertices and the added points in the subdivided mesh shown on the right. This figure is adapted from [177].

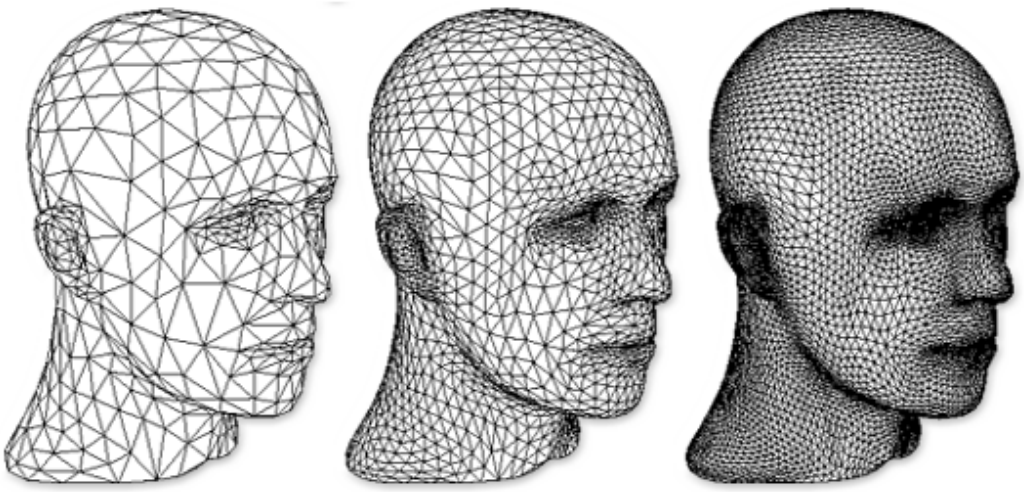

Figure 5.2: Application example of (two iterations of) Loop subdivision. This figure is adapted from [177].

We propose to replace the role of mesh connectivity in subdivision with intrinsic point neighbourhood information and formulate a set of intrinsic meshless refinement and geometric averaging rules in the form of weighted geodesic centroids of these local neighbourhoods.

Fleishman et al. [49] and Guennebaud et al. [62,63] touch upon the notion of meshless surface subdivision. In [49], the authors generate progressive level-of-details of point clouds by transferring the mesh-based idea of subdivision displacement maps to the point cloud case. They devise a purpose-built point cloud simplification method for the generation of a base point set and present both a projection and a local, uniform upsampling operator with the help of local surface approximation using MLS [5] (Section 4.3.1). The authors successfully mimic the principle of mesh-based subdivision displacement mapping for surfaces in point cloud form but do not consider the idea of meshless surface subdivision.

Similarly, Guennebaud et al. [62, 63] are concerned with the upsampling of "surfel sets", i.e. input points equipped with normal, local sampling density (surfel radius) and texture information, for magnified point rendering by splatting. Their interpolatory method requires that the underlying point cloud is regularly uniformly distributed and noise-free and that features such as crease lines have been detected and adequately sampled in a pre-processing step. Their method is restricted in applicability to surfaces in $\mathbb{R}^3$ and is intended for the operation on top of a splatting algorithm such as Zwicker et al. [179,180] providing the surfel information. As is typically the case for independently determined neighbourhoods such as the authors' enhanced k nearest neighbours-based polygonal fan neighbourhoods, their (extrinsic) proximity and refinement operators are affected by over-

lapping neighbour relations (Section 2.4.1) and the need for refinement rules varying with the number of neighbours of the point under consideration. This interpolatory mesh subdivision-inspired method is used for the locally smooth upsampling of surfel sets as part of a dynamic splatting algorithm. The more general notion of meshless surface subdivision is not considered.

By contrast, this chapter is concerned with the notion of intrinsic meshless surface subdivision and makes no assumptions on the availability of normal or local density information or the regularity of the input point cloud. The intrinsic natural neighbourhood concept of Section 2.4.3 is used to avoid the problem of overlapping neighbour relations. The concept is not restricted to surfaces in $\mathbb{R}^3$ but extends to higher dimensions. Inspired by recent work on geodesic curve subdivision by Wallner and Dyn [164–166], it offers a theoretical basis for global convergence and smoothness analysis. The intrinsic meshless surface subdivision framework is presented in the following section.

5.2 Intrinsic meshless surface subdivision

Subdivision schemes incorporate refinement and geometric averaging rules in the form of weighted averages of local neighbourhoods. Mesh-based subdivision schemes are derived in a parametric setting ignoring the geometric embedding of the points in space. As a result, they are formulated in terms of local mesh connectivity rather than object geometry. In the following, local neighbourhood relations are determined from intrinsic point proximity information instead using intrinsic natural neighbourhoods.

I start this section with the presentation of our intrinsic meshless subdivision scheme. This is followed by the discussion of our new method for the computation of geodesic centroids on manifolds, which is at the heart of this scheme.

5.2.1 An intrinsic meshless surface subdivision scheme

Within our intrinsic meshless surface subdivision framework, weighted geodesic centroids of intrinsic natural neighbourhoods are used to define meshless surface subdivision rules. More specifically, we propose the following set of rules to be applied at each iteration:

Refinement rule: For each neighbour $q_j \in \mathcal{N}_p$, consider the union of intrinsic neighbours of $p, q_j \in P_l$, $\mathcal{N}_{pq_j}$. Upsample P_l by inserting the weighted geodesic centroid, $c(\mathcal{N}_{pq_j}) \in P_{l+1}$, of $\mathcal{N}_{pq_j}$.

Geometric averaging rule: Replace $p \in P_l$ by the weighted geodesic centroid, $c(\mathcal{N}_p) \in P_{l+1}$, of its intrinsic Voronoi neighbourhood $\mathcal{N}_p$.

This use of weighted centroids in the refinement and geometric averaging rules is reminiscent of both classical subdivision schemes [177] and the repeated averaging approach towards the generation of subdivision surfaces [89, 121]. These subdivision rules are incorporated into our meshless surface subdivision algorithm as summarised in Algorithm 6.

In its initialisation phase, the algorithm buckets the input point set P_l in a Cartesian grid, subsequently used to support weighted intrinsic distance mapping and weighted geodesic centroid computation. Initialisation is completed with the computation of the (discrete) geodesic Voronoi diagram of P_l, $VD(P_l)$. Main processing loops over all points in P_l and proceeds with the upsampling of P_l to P_{l+1} using joint neighbourhood information readily available from $VD(P_l)$ and our refinement rule. Following this refinement step, $VD(P_{l+1})$ is computed. Intrinsic meshless surface subdivision is concluded with the application of our geometric averaging rule to each point in the refined point set P_{l+1} yielding the final, subdivided point set.

The applicability of this subdivision algorithm does not depend on a preceding simplification step. Potential algorithm applications, however, include the case in which it may be desirable to simplify an excessively dense point cloud P to a suitable base point set P_0 in the expectation that recursive meshless subdivision of P_0 results in a smoother, more regularly uniform and more compact approximation of the underlying surface than given by P. Since our subdivision algorithm requires the computation of geodesic centroids across the base point set P_0, for these centroids to be well-defined, any simplification of P needs to be performed subject to a minimum point density in P_0. Due to the method's simple control of a guaranteed point density, its purely intrinsic operation, its close relationship with the intrinsic natural neighbourhood concept and its efficient implementability, the intrinsic meshless surface simplification algorithm of Chapter 4 is used for the simplification of P to a base point set P_0 still sufficiently dense to support the computation of geodesic centroids. As another result of this pre-processing step, the geodesic Voronoi diagram of P_0 becomes available so that it does not need to be computed in the meshless subdivision algorithm's initialisation phase and the natural neighbours of a point $p_i \in P_0$ are readily known.

By performing the averaging intrinsically, our meshless subdivision rules raise the question

Input: Point cloud $P_l \in \mathbb{R}^m$.
Output: Subdivided point cloud P_{l+1}.

---

0 *** *INITIALISATION* ***
1 Bucket the base point cloud P_l in a m-dimensional Cartesian grid;
2 Compute the discrete geodesic Voronoi diagram, $VD(P_l)$, of P_l.
3
4 *** *MAIN PROCESSING* ***
5 **FOR** each point $p_i \in P_l$;
6 Determine the intrinsic Voronoi neighbourhood $\mathcal{N}_{p_i}$ from $VD(P_l)$;
7 **FOR** each neighbour $q_j \in \mathcal{N}_{p_i}$;
8 Determine the union of intrinsic Voronoi neighbours $\mathcal{N}_{p_i q_j}$ from $VD(P_l)$;
9 Compute the weighted geodesic centroid $c(\mathcal{N}_{p_i q_j})$;
10 (*Refinement rule*) Upsample P_l to P_{l+1} by inserting $c(\mathcal{N}_{p_i q_j})$;
11 **ENDFOR**
12 **ENDFOR**
13 Compute $VD(P_{l+1})$;
14 **FOR** each point $p_i \in P_{l+1}$;
15 Determine the intrinsic Voronoi neighbourhood $\mathcal{N}_{p_i}$ from $VD(P_{l+1})$;
16 Compute the weighted geodesic centroid $c(\mathcal{N}_{p_i})$;
17 (*Geometric averaging rule*) Replace p_i in P_{l+1} with $c(\mathcal{N}_{p_i})$;
18 **ENDFOR**

Alg. 6: One iteration of intrinsic meshless surface subdivision in pseudocode.

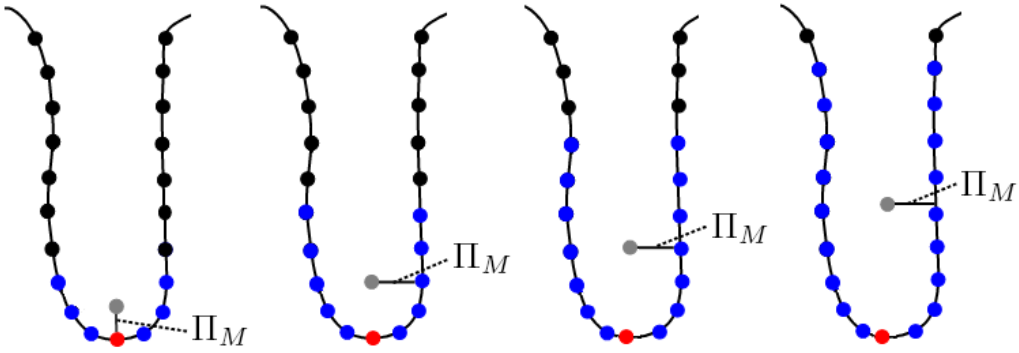

Figure 5.3: The unweighted centroid of a (blue) subset of this set of points is expected to be located on or near the underlying surface. Due to the use of intrinsic distances, this is the case when computing the geodesic centroid (red). By contrast, in the case of the Euclidean averaging of the (blue) points, the resulting centroid (grey) is located away from the underlying surface. This effect gets more pronounced when increasing the size of the subset (*from left to right*). Note that for geodesically close neighbourhoods and those kinds of neighbourhoods only, the orthogonally projected ($\Pi_M : \Omega_M^r \to M$) Euclidean average, i.e. the extrinsic mean, generally provides a good (first) estimate of the geodesic centroid (*left*).

of how to compute geodesic centroids on manifolds. Buss and Fillmore [24] present an algorithm for the computation of geodesic means on spheres. We generalise the underlying, earlier idea, see. e.g. Karcher [79] and references therein, of minimising a least squares

expression in geodesic distances in the following section.

5.2.2 Computation of geodesic centroids on manifolds

The benefit of performing the averaging intrinsically is that it ensures that subdivision generates smoother, denser representations which remain geometrically close to the surface. This is not guaranteed to be the case when considering Euclidean instead of geodesic centroids. For the simple example illustrated in Figure 5.3, Euclidean averaging ignores the non-linear, intrinsic geometry of the object and moves the centroid away from the surface. By contrast, since the computation of the geodesic centroid is based on intrinsic rather than Euclidean distances, it is inherently geometry-sensitive and falls onto the surface in each case.

The weighted geodesic centroid of a set of n points is defined as the point $g \in M$ which minimises the weighted sum of squared intrinsic distances to each point

$$J(g) := \frac{1}{2}\sum_{k=1}^{n} w_k d_M(g, p_k)^2,$$

where $w_1, \ldots, w_n$ represent the point weights, with $0 \leq w_k \leq 1$, $\sum_{k=1}^{n} w_k = 1$. In general, $\text{argmin}_g J(\cdot)$ may not exist or may not be a single point. However, if $p_1, \ldots, p_n$ are all contained in a sufficiently small open geodesic ball B_M on M, a unique solution, g_{B_M} of $J(\cdot)$, which happens to lie in B_M [79], is guaranteed. The property we are alluding to here is (geodesic) convexity, i.e. for any $p_i, p_j \in B_M$, the minimal geodesic from p_i to p_j is unique in M and contained in B_M.

In the Euclidean case, direct differentiation of $J(\cdot)$ yields the minimiser $g_E = \sum_{k=1}^{n} w_k p_k$. This simple result does not extend to the general case considered here but we can prove that any minimiser must satisfy:

$$V(g) := \sum_{k=1}^{n} w_k \nabla_M \frac{1}{2} d_M(g, p_k)^2 = 0 \tag{5.1}$$

Then, starting from a *good* initial guess g_0, we can track the minimiser g using back propagation with velocity field $V(\cdot)$. This is due to the fact that if $g_0 \in B_M$ and B_M as above, then $-V(x)$ points towards g_{B_M}, for $x \in B_M$ [79].

In practise, we set $g_0 = \Pi_M\left(\sum_{k=1}^{n} w_k p_k\right)$, where $\Pi_M : \Omega_M^r \to M$ is the orthogonal projection operator.[1] We now show that in the light of the considerations presented

[1] Alternatively, instead of g_0, we may start from any of the points p_k.

above, this extrinsic mean represents either a reasonable initial condition for the back propagation or a first approximation to the geodesic centroid. By a simple application of Lemma 17 of Wallner and Dyn [164], we have that

$$\left\| \sum_{k=1}^{n} w_k p_k - \Pi_M \left(\sum_{k=1}^{n} w_k p_k \right) \right\| \leq C(diam(B))^2,$$

where C is a global constant which depends on the curvatures of M. Then, let $B = B_M(x, \epsilon)$, for some $x \in M$ and $\epsilon > 0$. Since $\|p_i - x\| \leq d_M(p_i, x) \leq \epsilon$, we also have $\| \sum_{k=1}^{n} w_k p_k - x\| \leq \epsilon$. Therefore, since $\|g_0 - x\| \leq \|g_0 - \sum_{k=1}^{n} w_k p_k\| + \| \sum_{k=1}^{n} w_k p_k - x\|$, we obtain

$$\|g_0 - x\| \leq C\epsilon^2 + \epsilon,$$

which implies $d_M(g_0, x) \leq \epsilon(1 + H\epsilon)(1 + C\epsilon)$, for another constant H depending on global metric properties of M [104]. We only care for a simplified bound

$$d_M(g_0, x) \leq E\epsilon.$$

Finally, let $\overline{\xi} > 0$ be the maximal $\xi > 0$ such that $B_M(x, \xi)$ is (geodesically) convex. Note that it is a fact that if $\xi \leq \frac{1}{2} \min\left(inj(M), \frac{\pi}{\sqrt{K}}\right)$, where $inj(M)$ is the injectivity radius of M and K bounds all sectional curvatures in M from above, then $B_M(x, \xi)$ is convex for any $x \in M$. See §7.6 and §7.7 in Chavel [28]. For such a $\overline{\xi} > 0$ and provided $\epsilon \leq \overline{\xi}/E$, and $\{p_1, \ldots, p_n\} \subset B_M(x, \epsilon)$ for some $x \in M$, $g_0 \in B_M(x, \overline{\xi})$ and -$V(g_0)$ will be pointing towards g_{B_M}. Also, in case we want to use g_0 as an approximation to g_{B_M}, we have the (weak) bound $d_M(g_{B_M}, g_0) \leq (E + 1)\epsilon$. Therefore, g_0, as defined above, represents a sensible choice as the initial condition of an eventual back propagation step, or, in any case, a rough approximation to g_{B_M} with known error bound. See also Figure 5.3 (left). Note in particular that it is also a useful choice from the point of view of computational ease. The algorithm is summarised in Algorithm 7.

To demonstrate the applicability of this approach in the context of intrinsic meshless surface subdivision, we first consider the case of M representing the unit sphere in the following section.

5.3 Experimental results

We begin with the intrinsic meshless subdivision of a set of points sampled relatively regularly uniformly from the surface of the unit sphere in $\mathbb{R}^3$. This initial restriction

Input: Intrinsic Voronoi neighbourhood $\mathcal{N}_p$ of point $p \in P_l$. Weights w_i at points $q_i \in \mathcal{N}_p$.
Output: Weighted geodesic centroid g.

---

0 *** *Computation of extrinsic centroid* g_0 ***
1 Compute the Euclidean weighted centroid g_E of $\mathcal{N}_p$;
2 Compute g_0 by orthogonally projecting g_E;
3
4 *** *Computation of intrinsic centroid* g ***
5 Compute local weighted intrinsic distance maps $T_{\Omega_p^r}(q_i, \cdot)$ from each neighbour $q_i \in \mathcal{N}_p$ outwards and accumulate their squared values at the grid vertices;
6 Approximate the gradient of the accumulated distance maps using finite difference approximation;
7 Back propagate from g_0 towards g by following the negative gradient;

Alg. 7: Procedure for the computation of a weighted geodesic centroid in pseudocode.

to spherical geometry allows for the computation of precise geodesic distances without the need for numerical techniques. This way qualitative and quantitative aspects of our operator can be presented without the influence of the particular projection and gradient descent techniques utilised when processing more complex geometry. This presentation is followed by applications of our subdivision operator to more complex geometry.

To implement the geodesic centroid computation method, techniques for the computation of intrinsic distances between points on the surface, the projection of the starting point for the back propagation onto the surface and the computation of the back propagation itself are required. In the case of the unit sphere, these techniques are readily available and no numerical techniques are required. Intrinsic distances between points follow trigonometrically and orthogonal projection is trivial. Similarly, the exponential map and its inverse are directly available and may be used to implement the back propagation procedure. As a result, for the case of spherical geometry, our approach for geodesic centroid computation narrows down to the technique of Buss and Fillmore [24].

Our intrinsic meshless surface subdivision operator is applied to a base point set P_0 of 2144 points sampled relatively regularly uniformly from the unit sphere (Figure 5.7). The application of the subdivision operator to P_0 using the initial intrinsic natural neighbour information from $VD(P_0)$ yields the subdivided point set P_1 of Figure 5.8.[2] The result, P_2, obtained from the application of the operator to P_1 using intrinsic natural neighbourhood information from $VD(P_1)$ is shown in Figure 5.9. Given the relatively strongly regular uniformity of the distribution of the input data, uniform weighting was used for both the refinement and the geometric averaging rule in both iterations. The results produced

[2]The reconstruction of the surfaces from the point sets was performed with the help of PointsToPolys [130] throughout.

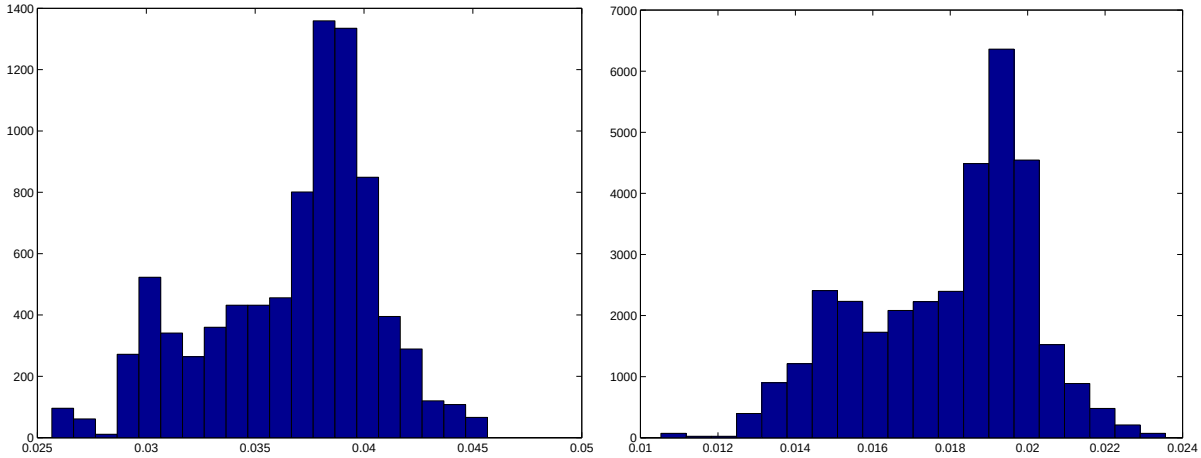

Figure 5.4: Histograms of the (spherical) distance from each point in P_1 (*left*) and P_2 (*right*) to its closest neighbour in the respective set. Both distributions feature pronounced modi and a relatively small range of distance values of approximately 0.025-0.045 and 0.0115-0.024 respectively. These values document the relatively regular uniformity of the point distributions generated by the two iterations of intrinsic meshless subdivision. The increase in point density observed for P_2 illustrates the (uniform) refinement effect of the subdivision iterations.

by the intrinsic meshless subdivision operator are presented alongside the vertex sets produced by the application of Loop subdivision[3] to a triangular mesh representation of P_0. As indicated by the detail views of Figure 5.9, the point distributions obtained from the two operators after two iterations are qualitatively similar with the slight non-uniformities in the distribution of P_2 being slightly more pronounced in the case of intrinsic meshless subdivision due to the use of uniform weights. There are, however, no noticeable differences in the smoothing effect of these two operators.

In order to analyse the point sets generated by our subdivision operator more quantitatively, the mean and the standard deviation of the distance from each point in the set to its closest neighbour(s) are computed for the subdivided point sets P_1 and P_2. For each p in the point set P_l, let $sd_k(p)$ denote the (spherical) distance from p to its kth closest neighbour. As an indicator for the uniformity of the density of point set P_l, consider $\rho(k) = \frac{\min_P sd_k(p)}{\max_P sd_k(p)}$. Since $\rho(k)$ represents an absolute measure, it may be too sensitive, therefore we compute instead $\widehat{\rho(k)} = \frac{\text{mean}_{(sd_k)} - \text{std}_{(sd_k)}}{\text{mean}_{(sd_k)} + \text{std}_{(sd_k)}}$, where mean and std stand for the mean and standard deviation of the spherical distances over the point set respectively.

The histograms of $sd_1(x)$ corresponding to the two sets of points are given in Figure 5.4. Note in particular in Table 5.1 that the values of $\widehat{\rho(k)}$, for $1 \leq k \leq 10$, are quite close to 1.0 therefore indicating small dispersion up to the 10th closest neighbour.

[3] I used the Loop subdivision implementation of Biermann and Zorin [16].

Model\k	1	2	3	4	5	6	7	8	9	10
P_1	0.807	0.832	0.832	0.821	0.788	0.809	0.8132	0.786	0.818	0.828
P_2	0.781	0.815	0.823	0.804	0.785	0.799	0.7980	0.784	0.810	0.822

Table 5.1: Values of the density uniformity measure $\widehat{\rho(k)}$ for P_1 and P_2 and with $k \in \{1, 2, \ldots, 10\}$. The values confirm the results of the histograms in Figure 5.4 and underline the relatively regular uniform distribution of the subdivided point sets. The increase in point density following the intrinsic meshless subdivision of P_1 expresses itself in the form of consistently reduced spherical distances of points in P_2 from their kth nearest neighbour.

Figure 5.10 presents an application example dealing with more complex geometry. The intrinsic meshless subdivision operator is applied to a base point set of 10088 points generated from the Michelangelo Youthful data set with the help of my intrinsic meshless surface simplification algorithm. Experimentation revealed the base point set distribution to be regularly uniform enough to allow for simple reciprocal distance weighting in the computations of the weighted geodesic centroids. The flatly shaded renderings of the surfaces reconstructed from the subdivided point sets P_1 and P_2 illustrate the smoothing effect of the intrinsic meshless subdivision. As indicated by the comparative illustrations in Figure 5.11, this approach may be used to obtain a smoother, more uniform and more compact representation of an highly dense point cloud. A similar effect is shown in Figure 5.12. The 50% decimated versions of the rocker arm and screwdriver CAD data sets available from the Cyberware website were meshlessly subdivided twice. The smoothing effect of these iterations is again apparent when comparing the surfaces reconstructed from the subdivided point sets to those reconstructed from P_0 and the non-simplified, non-subdivided data sets respectively.

The detail view of Figure 5.10 highlights the local clustering effect caused by overlapping neighbour relations as discussed in Section 2.4.1, an effect typically only encountered with neighbourhood concepts which determine the neighbour relations independently of each other. Our discrete approximation of the geodesic Voronoi diagram, however, implies discretisation error. As a result of such errors, the approximation accuracy is reduced. Since intrinsic natural neighbourhoods are determined from the geodesic Voronoi diagram approximation, there can be instances of points being assigned incorrect neighbour relations or none at all. The limited number of these instances encountered with the application examples presented here is considered preferable to the complications associated with addressing the problem of overlapping neighbour relations when using neighbourhood concepts which are not based on surface partitions such as the geodesic Voronoi diagram. See, e.g. Guennebaud et al. [62, 63].

Iteration 1

	$\Delta x(=\Delta y=\Delta z)$	r	$\|P_0\|$	$\|P_1\|$
Sphere	0.25	0.8	2144	8570
Youthful	1.0	2.2	10080	39888
Screwdriver	0.5	1.0	13577	56220
Rocker arm	1.0	1.9	20088	81442
Isis	0.1	0.2	187644	760162

Iteration 2

	$\Delta x(=\Delta y=\Delta z)$	r	$\|P_1\|$	$\|P_2\|$
Sphere	0.25	0.75	8570	36442
Youthful	0.25	0.75	39888	208010
Screwdriver	0.25	0.6	56220	295110
Rocker arm	0.25	0.45	81442	488212

Table 5.2: Parameter settings and point set sizes for the application examples; $\Delta x, \Delta y, \Delta z$ refer to the grid spacing in the three principal Cartesian grid directions; r represents the constant offset band radius.

The limited impact of these assignment errors is further illustrated by the detail view of Figure 5.13. The Isis point set was subdivided once with the regular uniformity of the distribution of the subdivided point set being only mildly affected by erroneous neighbourhood assignments. Unsurprisingly, given the density of the point set and in contrast to the processing of the Michelangelo Youthful base point set, intrinsic meshless subdivision of the Isis point cloud using geodesic centroids was no longer found to yield results significantly different from the use of subdivision by extrinsic centroids.

Table 5.2 summarises the parameter settings and point set sizes for the various application examples. Using my implementation, the offset band computation pre-processing step and an intrinsic meshless subdivision iteration took a maximum of a few hundred seconds each on a Pentium 4 2.8GHz machine with 512MB of memory running under MS Windows XP. Both offset band and geodesic Voronoi diagram computation efficiency generally depend strongly on the offset band radius r and the grid spacing, our settings of which are presented in the table. Details on various aspects of the implementation are provided in the following section.

5.4 Implementation details

The algorithm was implemented in C++ (Microsoft Visual C++ 7.1) with the help of the "Blitz++ 0.7 Numerical Library" [17]. The implementations of the offset band computation and weighted intrinsic distance mapping have already been discussed in Section 2.5.2. The computation of geodesic Voronoi diagrams on point set surfaces was discussed in Section 2.5.3. In the context of this application, the initial geodesic Voronoi diagram, $VD(P_0)$, is either computed during the algorithm's initialisation phase or follows directly from prior intrinsic meshless surface simplification (Chapter 4). My implementations of intrinsic natural neighbourhood computation (Section 5.4.1), geodesic centroid computation (Section 5.4.2) and orthogonal projection (Section 5.4.3) are discussed in the following.

5.4.1 Intrinsic natural neighbourhoods

As regards the computation of intrinsic natural neighbourhoods, the simultaneous computation of intrinsic distance maps across the surface during geodesic Voronoi diagram computation also allows for the Voronoi edges of $VD(P)$ and thus each point's intrinsic Voronoi neighbours to be obtained *during front propagation* as loci of intersection between the geodesic offset curves [33]. To see this, note that the set of (CLOSE) vertices on the min-heap gives the current positions of the various fronts. The loci of intersection are given by those vertices at which pairs of fronts originating from $p_i, p_j \in P, p_i \neq p_j$ meet, i.e. where $T_M(p_i) = T_M(p_j)$ at or near the vertices. Simple multilinear interpolation of the distance maps can be used if precise positions of the points of intersection of $T_M(p_i)$ and $T_M(p_j)$ are required.

To implement this idea, every time a grid vertex is set ALIVE, it is indexed with the vertex it was computed from, i.e. it is indexed with the source point of the propagation and thereby associated with its intrinsic Voronoi region. Then note that, for example, for the 3D point cloud case, the solution of the finite difference gradient approximation (2.3) during arrival time updating depends on at most three ALIVE neighbours of the grid vertex (Section 2.5.1). For a grid vertex to be located on a Voronoi edge, two of these neighbours have to belong to different intrinsic Voronoi regions as indicated by their indices. If this is the case, the pair of indices represents a pair of intrinsic Voronoi neighbours. Since all the fronts are computed simultaneously rather than sequentially, the vertices on or near the edges of the geodesic Voronoi diagram are detected during front propagation and thus without the need for any additional vertex visits which would

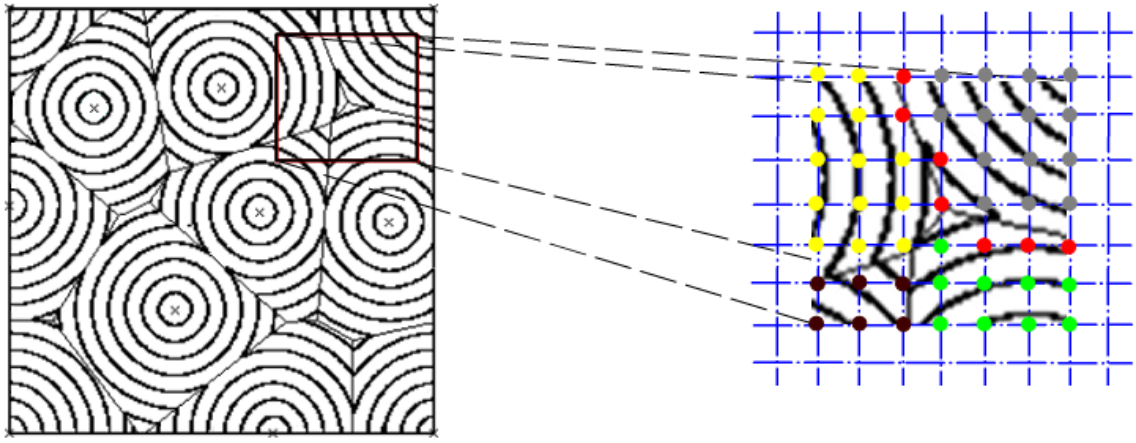

Figure 5.5: Detection of intrinsic natural neighbours during front propagation: The intrinsic Voronoi region membership of a grid vertex is indicated by colours grey, yellow, green and brown respectively. Since each vertex maintains this information, those (red) grid vertices on or near the Voronoi edges can be identified as part of the computation of their arrival times from their neighbouring (ALIVE) grid vertices during Fast Marching. The source points whose intrinsic Voronoi regions are sharing these edges represent intrinsic natural neighbours which are thus detected during front propagation. Any additional grid traversal is not required.

affect the algorithm's efficiency (Figure 5.5). A grid vertex located on an edge of the geodesic Voronoi diagram is assigned to the intrinsic Voronoi region of the neighbouring grid vertex used in the update step which features the smaller arrival time.

Since the same Voronoi edge will be detected multiple times, the corresponding neighbourhood relation is stored in a neighbourhood lookup table and any subsequent detection of the relation is ignored.

The resulting neighbourhood information is used in the context of geodesic centroid computation, my implementation of which is presented next.

5.4.2 Geodesic centroid computation

My implementation of Algorithm 7 (reproduced in Algorithm 8), proceeds from the computation of the weighted Euclidean centroid, $g_E = \sum_{k=1}^{\|\mathcal{N}_p\|} w_k p_k$, of $\mathcal{N}_p$, whose weighted geodesic centroid is to be computed (line 1). With the w_k being determined experimentally, g_E is readily computable. Since the normal estimated as part of MLS is not guaranteed to be orthogonal to the underlying surface [3, 11], I use an orthogonal projection operator introduced by Alexa and Adamson [3] and discussed in the following section, to project g_E onto M thereby obtaining g_0 (line 2).

Input: Intrinsic Voronoi neighbourhood $\mathcal{N}_p$ of point $p \in P_l$. Weights w_i at points $q_i \in \mathcal{N}_p$.
Output: Weighted geodesic centroid g.

---

0 *** *Computation of extrinsic centroid* g_0 ***
1 Compute the Euclidean weighted centroid g_E of $\mathcal{N}_p$;
2 Compute g_0 by orthogonally projecting g_E;
3
4 *** *Computation of intrinsic centroid* g ***
5 Compute local weighted intrinsic distance maps $T_{\Omega_p^r}(q_i, \cdot)$ from each neighbour $q_i \in \mathcal{N}_p$ outwards and accumulate their squared values at the grid vertices;
6 Approximate the gradient of the accumulated distance maps using finite difference approximation;
7 Back propagate from g_0 towards g by following the negative gradient;

Alg. 8: Procedure for the computation of a weighted geodesic centroid in pseudocode.

My implementation of Fast Marching for surfaces in point cloud form (Section 2.5.2) is then used to compute weighted intrinsic distance maps from each neighbour in $\mathcal{N}_p$ outwards. The squared distance map values are accumulated at the grid vertices visited during these front propagations. This approximates $d_M(g, p_k)^2$ in (5.1) for all $p_k \in \mathcal{N}_p$. To avoid unnecessary propagation, the extent of the distance mapping is limited to the radius of the sphere enclosing $\mathcal{N}_p$ (line 5).

To compute the intrinsic gradient of the distance map values accumulated at the grid vertices, the normalised central difference operator of Blitz++ [17] is used (line 6). Following this gradient approximation, (5.1) can be evaluated: A standard Runge-Kutta gradient descent process with multilinear interpolation is used to back propagate from g_0 towards g by following the (negative) gradient of the accumulated distance maps (line 7). Runge-Kutta represents a widely-used and well-understood procedure and I refer to Press et al. [131] for details on its implementation.

This completes the computation of the weighted geodesic centroid of neighbourhood $\mathcal{N}_p$. My implementation of the orthogonal projection operator used in this context is presented in the following section.

5.4.3 Orthogonal projection

The normal estimated as part of the first step of MLS surface approximation is frequently used for orthogonal projection purposes. See, e.g. Pauly et al. [125]. Since MLS has already been implemented in the context of my intrinsic meshless surface simplification algorithm, it may seem useful to employ this approach here as well. However, as pointed

Input: Point $p_i \in P$. Approximation error threshold ϵ.
Output: Approximately orthogonal projection p_i' of p_i.

```
0 *** Initialisation ***
1     Set p_i' = a(p_i);
2
3 *** Projection ***
4     REPEAT
5         Compute n at p_i' and a(p_i');
6         Set p_i' = p_i − (n^T(a(p_i') − p_i)n;
7     UNTIL ‖n^T(a(p_i') − p_i')‖ < ε;
```

Alg. 9: Approximately orthogonal projection algorithm in pseudocode.

out by Alexa and Adamson [3] and Amenta and Kil [11], the first weighted least squares regression of MLS fits a support plane to a point which will frequently be located close to but not on the underlying surface. In these cases, the support plane is therefore not tangent to but near the surface and its normal will generally not coincide with the surface normal. Since an orthogonal projection procedure is required in the context of geodesic centroid computation, an orthogonal projection operator put forward by Alexa and Adamson [3] is therefore adopted here instead.

Although in the case of Alexa and Adamson's [3] projection procedure, the normal is again estimated using weighted least squares, the normal estimate and projection are iteratively improved subject to a user-defined approximation error threshold. More specifically, their projection method repeatedly projects a point p_i onto local support planes defined by the normal estimate n at p_i and the weighted average

$$a(p_i) = \frac{\sum_{j=1}^{\|P\|} \theta_j(\|p_i - p_j\|)p_j}{\sum_{j=1}^{\|P\|} \theta_j(\|p_i - p_j\|)}.$$

The weight function θ_j determines the influence of point p_j and is assumed to be smooth, positive and monotonically decreasing in distance. Thus, similar to MLS, Alexa and Adamson's [3] method is based on an implicit surface definition and operates meshlessly. The implicit surface approximation is given by those points p_i for which $f(p_i) = n^T(a(p_i) - p_i)$ equals zero, i.e. $f(p_i)$ describes the distance of the weighted point average $a(p_i)$ in normal direction from the implicit surface described by $f(p_i) = 0$.

The orthogonal projection procedure then consists of the steps summarised in Algorithm 9. Thus, at each iteration, the current projection p_i' is used to update n and $a(p_i')$, i.e. the position of the support plane (line 5); p_i is then projected onto this support plane to obtain an updated projection approximation p_i' (line 6) (Figure 5.6). Provided this

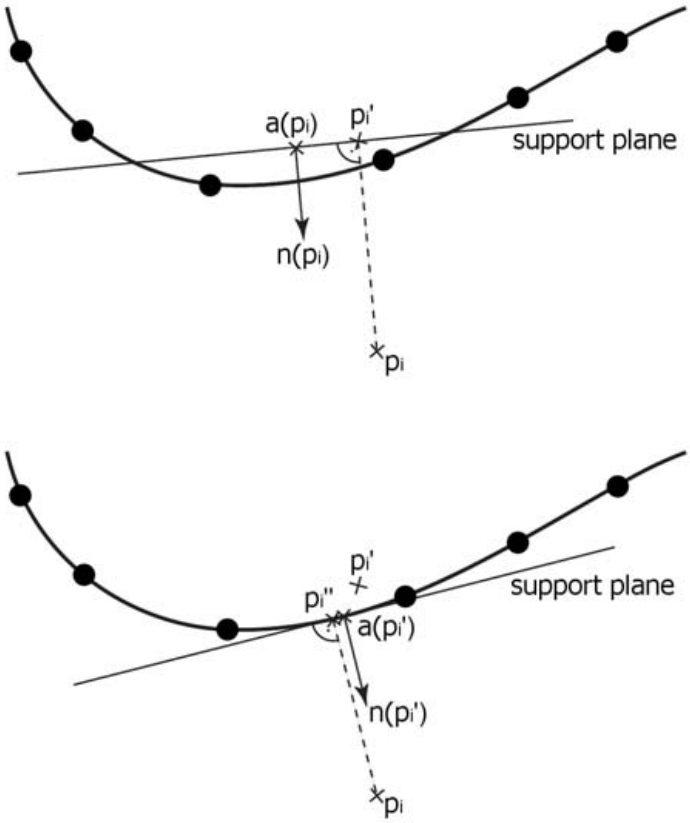

Figure 5.6: Approximately orthogonal projection of point p_i onto a surface: The current projection p_i' of p_i (*top*) is iteratively improved by fitting a local support plane using $a(p_i')$ and $n(p_i')$; p_i is then projected onto this plane to obtain the new approximation (*bottom*). The figure shows the first two steps of the process and is adapted from [3]. The view is in tangent direction.

process converges, it continues until p_i' is within a given ϵ of the implicit surface, i.e. $\|n^T(a(p_i') - p_i')\| \approx 0$ (line 7). Thus, in contrast to MLS, this technique guarantees (almost) orthogonal projection onto the surface since $p_i' - p_i$ is enforced to be in the direction of normal vector n at p_i'.

As regards the implementation of this technique, the normal estimation process coincides with the method used in the context of MLS surface approximation (Section 4.3.1). During this processing step, my code also computes the weighted average $a(p_i)$, again using a Gaussian as weight function θ_j. The projection of p_i onto the corresponding support plane is performed using a simple dot product computation in newmat11 [117]. These steps are repeated until the termination condition (line 7) is met for a user-controlled ϵ. If the algorithm fails to converge, the closest approximation is returned instead.

This completes the discussion of the implementation aspects of the intrinsic meshless surface subdivision scheme. The following section summarises and discusses the results presented in this chapter.

5.5 Summary and discussion

This chapter introduced the concept of intrinsic meshless surface subdivision based on the computation of weighted geodesic centroids on manifolds represented by point clouds. This technique was implemented and its applicability shown using a new method for the computation of such centroids. The main contributions of this chapter may thus be summarised as

- the introduction of the notion of intrinsic meshless surface subdivision.
- the introduction of a new method for the computation of geodesic centroids on manifolds.
- the introduction of the first intrinsic meshless surface subdivision scheme and
- the implementation and application of this scheme.

The consideration of local intrinsic point proximity instead of mesh connectivity helps to overcome some of the limitations of existing mesh-based surface subdivision schemes. For example, by working with the raw data and operating directly across the point cloud, problems associated with the complexity of the topological subdivision of high-dimensional meshes can be avoided.

With regard to the application examples given, the choice of weights was guided by experimentation rather than by theoretical evidence for our scheme's convergence towards a smooth limit surface. Depending on the extent of any non-uniformities in the distribution of the input points, this experimental selection of point weights tends to be elaborate. The issue of selecting the point weights more systematically is closely related to the theoretical analysis of our intrinsic meshless surface subdivision scheme. This is commented on in the following, concluding chapter.

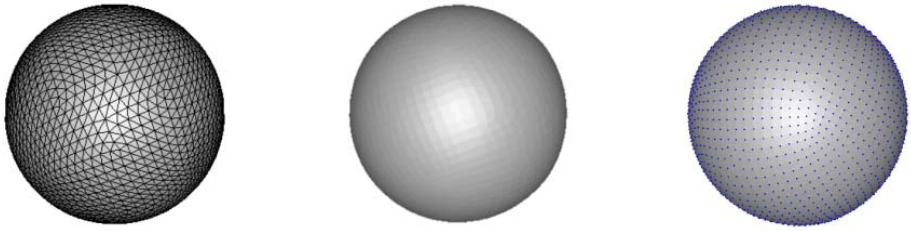

Figure 5.7: Relatively regularly uniformly distributed base point set, P_0, of 2144 points acquired from the unit sphere (*right*). The triangular base mesh generated from P_0 for the support of Loop subdivision is shown on the left. A flatly shaded rendering of the reconstructed surface is shown in the centre.

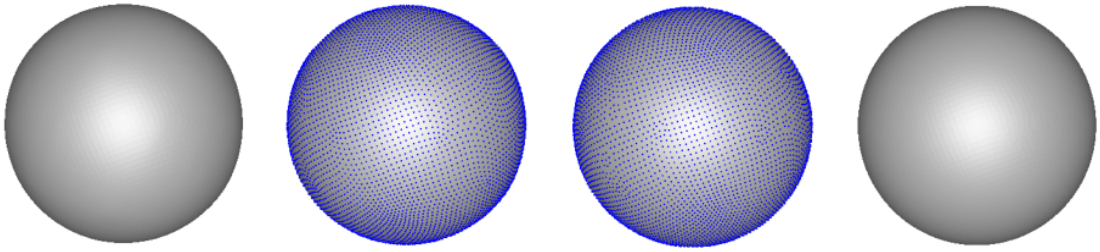

Figure 5.8: Results after one iteration of Loop (*left*) and intrinsic meshless subdivision (*right*); $\|P_1\| = 8570$ points in both cases. Flatly shaded renderings of the reconstructed surfaces are shown next to each point set.

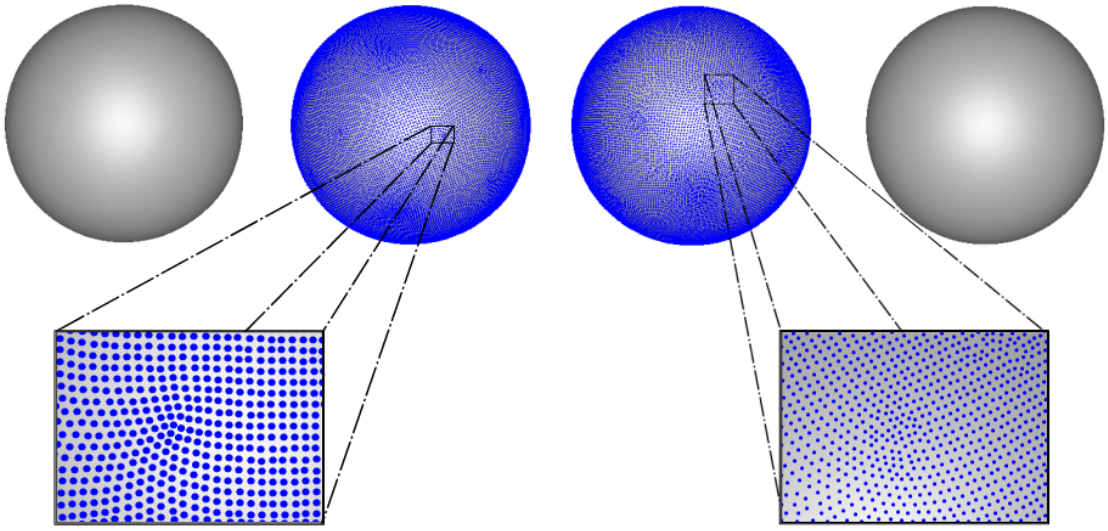

Figure 5.9: Results after the second iteration of Loop (*left*; 34275 points) and intrinsic meshless subdivision (*right*; 36442 points). The corresponding reconstructed surfaces are given next to each point set. The detail views indicate how the slight non-uniformities in the distribution of P_0 lead to slightly more pronounced local non-uniformities in the case of intrinsic meshless subdivision due to the use of uniform weighting. These slightly more pronounced non-uniformities do not have any noticeable effect on the smoothness of the surface.

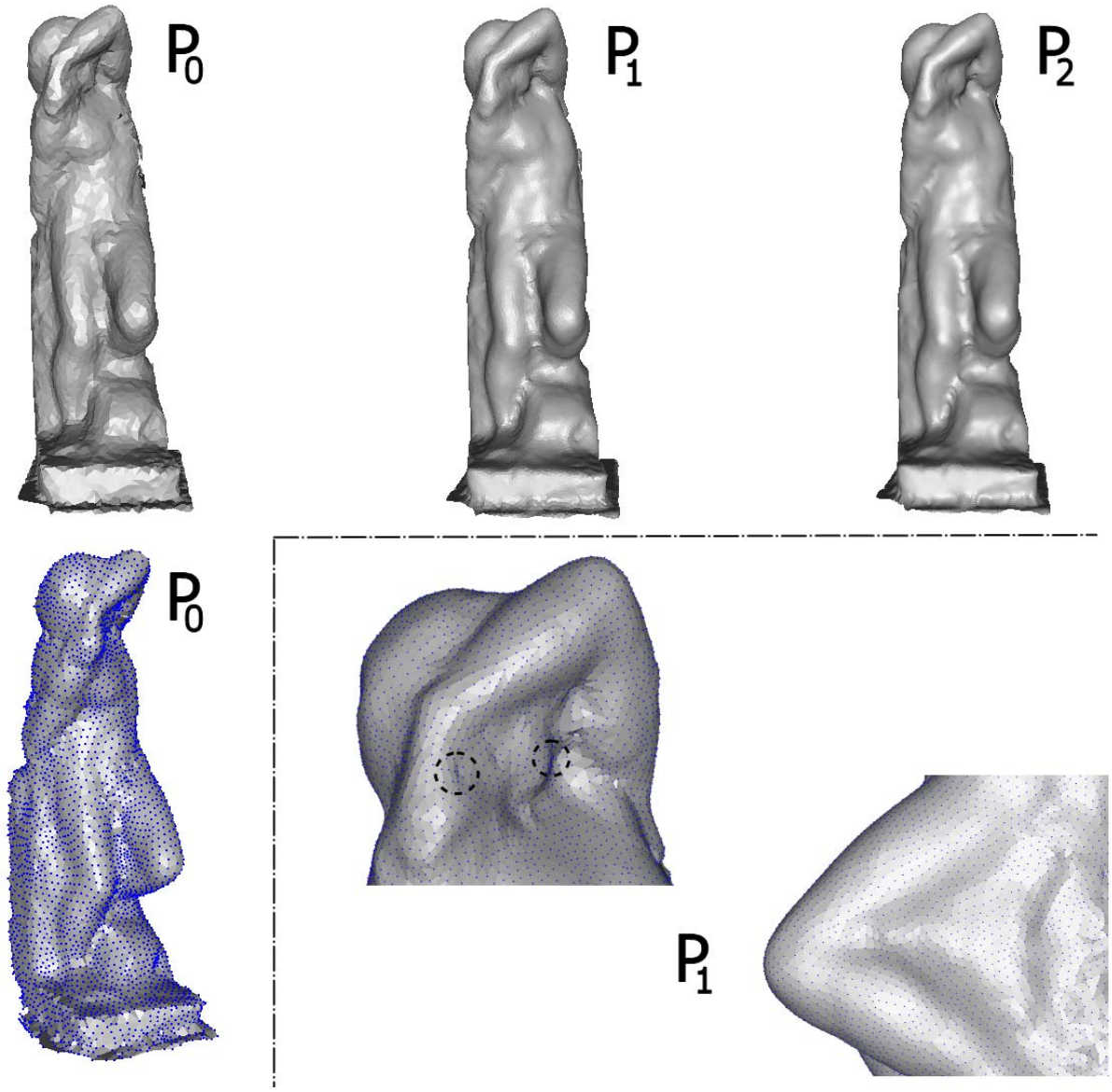

Figure 5.10: Flatly shaded renderings of the surfaces reconstructed from point sets $\|P_0\| = 10088$ (*top and bottom left*), $\|P_1\| = 39888$ (*centre and bottom right*) and $\|P_2\| = 208010$ generated from the Michelangelo Youthful data set. The smoothing effect of intrinsic meshless surface subdivision is clearly visible. As illustrated by the detail front and side views, the subdivided point sets are distributed relatively regularly uniformly. Instances of local non-uniformities (encircled in black) are caused by discretisation errors during discrete geodesic Voronoi diagram computation.

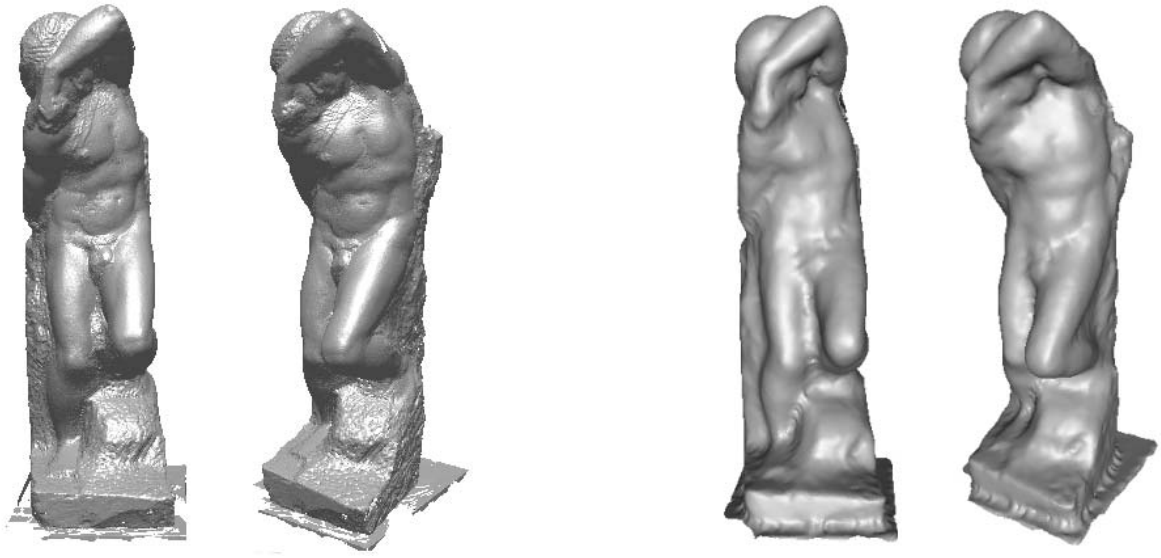

Figure 5.11: Flatly shaded renderings of the surfaces reconstructed from the Michelangelo Youthful point set (*left*) and P_2 of Figure 5.10 (*right*) illustrating how intrinsic meshless subdivision of a base point set resulting from intrinsic meshless surface simplification may be used to obtain a smoother, more regularly uniform and more compact representation of the original data set of 1728305 points.

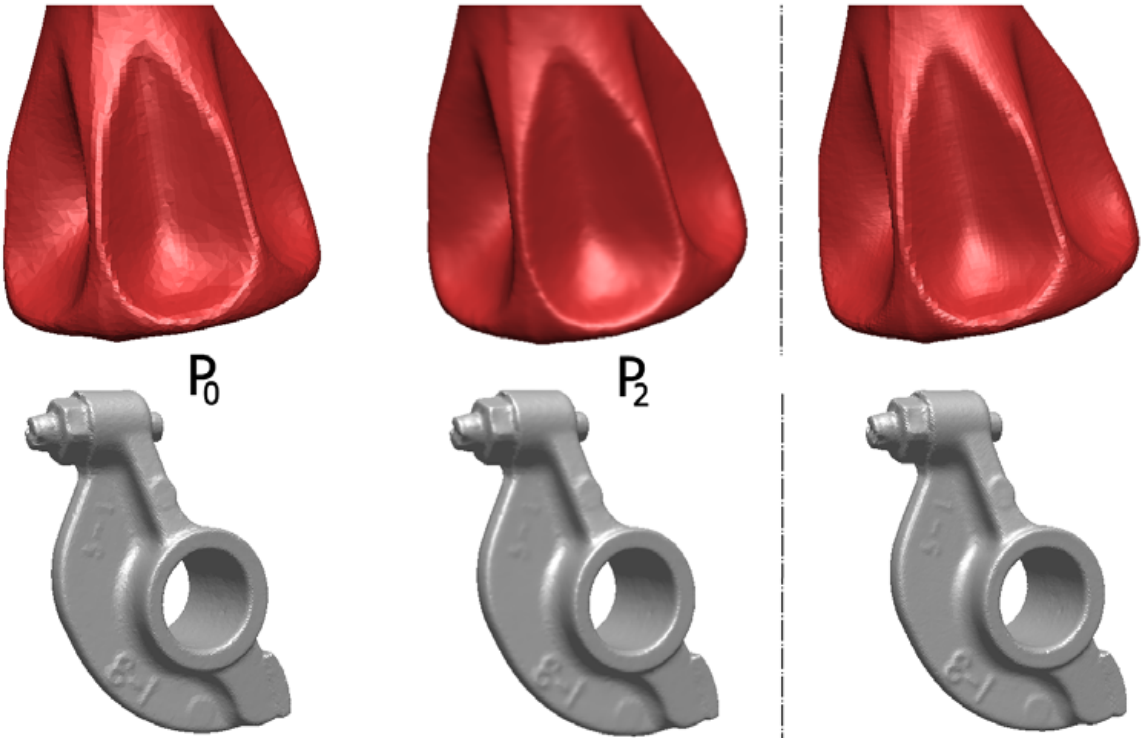

Figure 5.12: On the left, the flatly shaded renderings of surfaces reconstructed from the 50% decimated screwdriver (*top*) and rocker arm (*bottom*) CAD data sets (P_0) are shown. The reconstructed surfaces obtained from two iterations of intrinsic meshless subdivision of these point sets are presented in the centre (P_2). The surfaces reconstructed from the non-simplified, non-subdivided CAD point sets are given on the right. The smoothing effect of intrinsic meshless subdivision is again clearly visible.

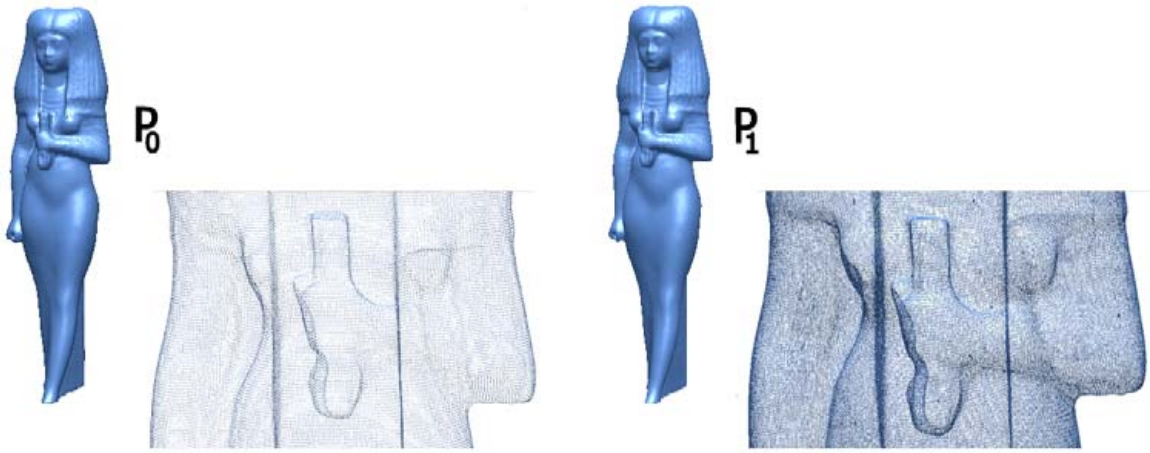

Figure 5.13: Point set detail views and smoothly shaded renderings of the Isis data set (*left*), P_0, $\|P_0\| = 187644$, and the subdivided point set P_1 (*right*), $\|P_1\| = 760162$, resulting from one iteration of intrinsic meshless subdivision. Due to the high density of P_0, the difference in location between the (uniformly weighted) extrinsic and the corresponding geodesic centroid was found to be negligible.

Chapter 6

Conclusion and future work

This thesis has been concerned with the development of algorithms for the intrinsic processing of surfaces in general and those represented in point cloud form in particular. It has been argued that the combination of intrinsic and point-based surface processing helps to overcome limitations of the traditional mesh-based approach when dealing with the highly complex point-sampled geometry typically produced today. This chapter summarises the principal contributions made in this context (Section 6.1), other results obtained (Section 6.2), and presents directions for future work (Section 6.3).

6.1 Principal contributions

As regards the intrinsic processing of surfaces in general, an *algorithm template for the intrinsic point sampling of surfaces* in the most widely-used forms of representation has been presented. It has been shown to enforce an user-controlled, guaranteed sampling density and to be equally well instantiable for planar domains and surfaces in triangular mesh and point cloud form. The design of the algorithm template is modular with the various instantiations constituting a library of modules for the very efficient uniform and flexibly non-uniform sampling of surfaces, operations of fundamental importance in a surface processing context. The modules' usefulness has been shown for images, triangular meshes and point clouds.

With respect to the processing of point-based surface representations, a first *intrinsic meshless surface simplification algorithm* has been proposed. It is based on the template instantiation for point clouds and allows for very efficient intrinsic uniform and non-uniform point cloud sub- and resampling. The absence of any combinatorial mesh data

structure permits the meshless simplification of massive point clouds in-core. The algorithm inherits the properties of the template, in particular the user-controlled, guaranteed sampling density and the inherently progressive operation. Local non-uniformities in the point cloud distribution or illegitimate holes of simple complexity can be addressed using the resampling method. Non-uniform simplification may be driven by measures such as local curvature or texture information or a combination thereof. Since the processing occurs intrinsically and meshlessly, the need for generally error-prone and prohibitively costly mesh pre-processing steps is avoided. Also, the dimensionality of the problem coincides with that of the surface rather than that of the embedding space.

An intrinsic neighbourhood concept for point-sampled geometry in the form of *intrinsic natural neighbourhoods* has been suggested. It is based on the geodesic Voronoi diagram of the point cloud under consideration. Unlike existing extrinsic neighbourhood concepts, intrinsic natural neighbourhoods therefore take the geometric embedding of the point cloud into account. As a result, the concept has been found to avoid a number of shortcomings associated with alternative extrinsic neighbourhood concepts.

The notion of *intrinsic meshless surface subdivision* has been introduced. This concept has been implemented in the form of the first *intrinsic meshless surface subdivision scheme* which makes extensive use of intrinsic natural neighbourhood information. Unlike mesh-based surface subdivision, intrinsic meshless surface subdivision operates directly across the surface and is therefore inherently geometric in nature. As in the case of intrinsic meshless surface simplification, the dimensionality of the problem coincides with that of the surface making the concept particularly attractive when dealing with high-dimensional surfaces or surfaces of high co-dimension. The restriction in applicability of mesh subdivision schemes to one particular type of mesh element and the need for costly pre-processing steps for the generation of a base mesh are also avoided this way. Our intrinsic meshless surface subdivision scheme has been shown experimentally to generate results qualitatively similar to mesh-based surface subdivision.

In summary, algorithms for the point sampling of surfaces, the collection of intrinsic point proximity information and the meshless simplification and subdivision of surfaces in point cloud form have been proposed. These algorithms operate truly intrinsically throughout. As a result, prohibitively costly and generally error-prone mesh pre-processing steps are avoided, the dimensionality of the problem coincides with that of the manifold and the geometry of the point set is no longer ignored. The algorithms are particularly suitable for the processing of complex point-sampled geometry as is typically produced today and which tends to push mesh-based processing beyond its limits.

6.2 Other results

As part of our intrinsic meshless surface subdivision scheme, we have introduced a *new method for the computation of geodesic centroids on manifolds*. Unlike neighbourhood centroids based on Euclidean distances, geodesic centroids are geometry-sensitive and take into account any non-linearities of the underlying manifold. Our algorithm for the computation of such centroids is based on the minimisation of a least squares expression in the geodesic distances between the point neighbours thereby generalising an earlier idea (see Karcher [79] and references therein) to the manifold case. This result is of interest in a number of fields other than computer graphics including, for example, intrinsic statistical shape analysis [50] and variational theory [75].

6.3 Future work

The research presented in this thesis has opened up a number of avenues for future work.

The intrinsic natural neighbourhood concept introduced in Chapter 2 has proved experimentally to be superior to existing extrinsic neighbourhood concepts for the purposes of this thesis. A theoretical study evaluating the properties of the various neighbourhood concepts is required to obtain more rigorous comparative results. These would facilitate the systematic selection of the best neighbourhood concept to use for the application at hand.

The intrinsic surface sampling algorithm of Chapter 3 places a single farthest point sample at a time. It would be interesting to consider the case of adding multiple samples per iteration to speed up the sampling process further. For example, rather than placing a single sample at the farthest Voronoi vertex, $n > 1$ samples could be placed at the n farthest Voronoi vertices at each iteration. Although the resulting sample sequence would no longer represent a farthest point sequence in the sense of Chapter 3 and the sampling density guarantee would no longer apply, the efficiency gain may prove a useful trade-off for particularly time demanding applications without strict sample density requirements.

The intrinsic meshless surface simplification algorithm of Chapter 4 and the intrinsic meshless surface subdivision algorithm of Chapter 5 require the computation of an adaptive offset band when dealing with non-uniformly distributed point clouds. The approach used in this thesis fits a set of offset balls. A more sophisticated method would fit an ellipsoidal rather than spherical offset band. This approach could allow for superior adaptation of the offset band to the local geometry and point density and thus result in a

smaller size of the band. An ellipsoidal offset band could be computed using, for example, iterative principal component analysis [104]. Also, studies of the performance of these algorithms when dealing with relatively noisy point sets or high-dimensional surfaces or surfaces of high co-dimension are left to be performed. For extremely high-dimensional data, intrinsic meshless surface subdivision operators should be devised which upsample the point-sampled geometry more conservatively than the operator suggested in Chapter 5.

The convergence and smoothness properties of the intrinsic meshless surface subdivision scheme are to be analysed theoretically. In this context, we intend to build upon ideas of Wallner and Dyn [164–166], in particular their convergence and smoothness analysis of non-linear geodesic curve subdivision by proximity to a corresponding linear extrinsic subdivision scheme.

As first proposed in the context of univariate spline subdivision schemes by Lane and Riesenfeld [89], the practical and theoretical aspects of repeating the geometric averaging step several times after each refinement step are also to be investigated. Using a higher number of averaging steps in each iteration of the intrinsic meshless surface subdivision process is expected to result in higher smoothness of the subdivided surface.

Appendix A

Glossary of terms

Characteristic curve — Curve along which a PDE is reduced to an ordinary differential equation. The characteristic line of a PDE is determined by the solution of its characteristic equations. These equations are solved as part of the method of characteristics for the solution of PDEs [74].

Co-dimension — Difference in the dimension of an object and the dimension of the embedding space containing it. For example, the co-dimension of a two-dimensional manifold in $\mathbb{R}^4$ is two.

Compact manifold — A manifold without boundary.

Complete manifold — A manifold with at least one complete metric which induces the manifold's topology. For example, a geodesically complete (Riemannian) manifold is a manifold for which geodesics can continue indefinitely, i.e. there exists no boundary nor any singularity which can be reached in finite time. In this case, there always exists at least one minimising geodesic between any two points on the manifold.

Connected manifold — A manifold M is connected if any two points in M can be connected by a curve lying completely within M.

Differentiable manifold — A smooth, i.e. infinitely differentiable (C^∞), manifold.

Eikonal equation — Equation of the type $\nabla T(x_1, \ldots, x_n)^2 = F(x_1, \ldots, x_n)^2$ describing the travel time propagation in an isotropic medium. The Eikonal equation was originally derived in the context of the modelling of wave-light propagation [145].

Extrinsic	Properties of a manifold depending on the chosen embedding. Extrinsic surface processing studies geometry from the point of view of a being living in the embedding space. For example, the extrinsic centroid of points on a manifold embedded in higher-dimensional Euclidean space is given by the centroid of the points in Euclidean space projected onto the manifold.
Geodesically convex	A manifold is geodesically convex if any two points in a neighbourhood on the manifold are connected by a unique geodesic contained within the neighbourhood.
Intrinsic	Properties of a manifold which do not depend on its particular embedding. Intrinsic surface processing studies geometry not from the point of view of the geometry's embedding space but rather from the view of an individual living on the manifold. For example, the intrinsic centroid of points on a manifold is given by the centroid of the points in their geodesic distances.
Irregular sampling	Stochastic sampling, i.e. sampling at non-deterministic distances.
Isotropic	Invariant with respect to direction.
Level-of-detail representation	Representation of a geometric object at multiple resolutions.
Manifold	Topological space characterised by the fact that each neighbourhood around a point on the manifold is topologically equivalent to a disk.
Manifold with boundary	A manifold with boundary is a manifold everywhere except in some places where it is topologically equivalent to a half-disk.
Non-uniform sampling	Sample set featuring a sample distribution varying non-uniformly across the manifold, i.e. the probability of a point being sampled depends on the value of an adaptivity measure.
Point sampling	Sample values obtained from the selection of individual points on the manifold.
Power spectrum	Fourier transform of a signal's autocorrelation function.
Progressive sampling	Sampling which at each stage of the sampling process provides a complete description of the manifold.
Regular sampling	Sampling with deterministic inter-sample distances. For example, sampling at the vertices of a regular grid.

Riemannian manifold	A manifold with a metric tensor, i.e. a tensor allowing to compute the distance between any two points in the given space. For a (geodesically) complete Riemannian manifold, the metric for a pair of points x, y on the manifold is given by the length of the minimal geodesic between x and y.
Sampling	A set of samples or the process of selecting sample points on the manifold.
Sectional curvature	Sectional curvature describes the curvature of a Riemannian manifold by the rate of geodesic deviation. More specifically, consider the space spanned by the geodesics of the manifold emanating from a point p. This space represents a sub-manifold with the induced metric [28]. It is thus a surface and the Gaussian curvature of this surface represents the sectional, or Riemannian, curvature of the manifold at p.
Surfel	"Surface element", i.e. a point equipped with normal, texture, depth, etc. information and used as a display primitive.
Uniform sampling	Sample set of approximately constant density, i.e. the probability of a point being sampled is equal for all points on the manifold.

Appendix B

Concepts from computational geometry

This appendix defines a number of concepts from computational geometry referred to repeatedly in this thesis. The presentation follows Amenta et al. [8] and Boissonnat and Cazals [20] and starts with the definition of the notion of the medial axis of a manifold.

Definition 1 The *medial axis* of a manifold M embedded in $\mathbb{R}^m$ is the closure of the set of points in $\mathbb{R}^m$ with more than one nearest neighbour on M [8].

Note that this definition includes elements exterior to a closed surface. An example for a medial axis in $\mathbb{R}^3$ is shown in Figure B.1. The shape of the medial axis necessarily reflects the proximity to parts of the surface. More specifically, the medial axis will be close to the surface where its curvature is high and farther from the surface in flat regions. This property is exploited in the context of the notions of local feature size and ϵ-sampling density defined next.

Definition 2 The *local feature size* at a point p, $LFS(p)$, on M is the Euclidean distance from p to (the nearest point of) the medial axis [8].

Definition 3 The set $P \subset M$ represents an *ϵ-sampling* of M if no point p on M is farther than $\epsilon \cdot LFS(p)$ from a point of P [8].

Thus, the sampling P represents an ϵ-sampling of M if the Euclidean distance from any point $p \in M$ to the nearest sample point is at most ϵ times the distance from p to the nearest point of the medial axis of M. Note that this definition implies that ϵ-samples are

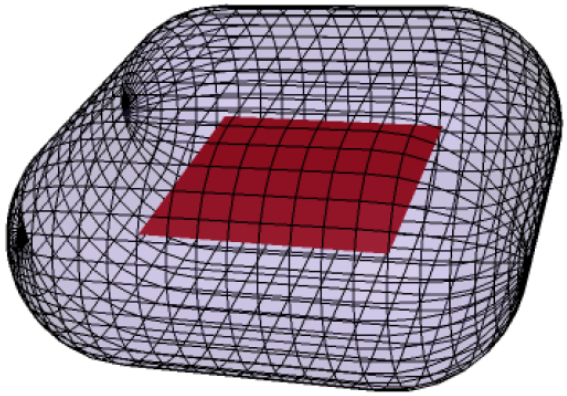

Figure B.1: Example for the (red) medial axis of a (rounded transparent) surface. Note that in the case of a non-convex surface, the medial axis would have parts located on the outside of the surface. This figure is adapted from [8].

placed more densely in regions of high curvature and less densely in flat surface regions. It also implies that at sharp corners, the sampling density would have to be infinite due to the medial axis touching the surface. Thus, either M should be smooth or sharp corners need to be dealt with heuristically.

For the ϵ-sampling condition to be controlled for in an application context, the medial axis needs to be computed. This can be challenging in practice [8]. As an alternative in the 3D case, Amenta et al. [8] propose to consider the distance to the nearest *pole* as a more easily computable estimate of a sample's distance to the medial axis. This idea exploits the fact that in the 3D case, many vertices of the Euclidean Voronoi diagram [120] are located near the medial axis. To filter out those vertices which do not lie near the medial axis, only those two vertices of the Voronoi region of a sample on M are selected as estimates of the location of the medial axis which are farthest from the sample on either side of M. These two vertices are called the poles of the sample. Figure B.2 illustrates this concept.

Further concepts related to the (3D) Euclidean Voronoi diagram and referred to in this thesis include the restricted (Euclidean) Voronoi diagram and its dual, the restricted (Euclidean) Delaunay triangulation, defined next.

Definition 4 The Voronoi diagram of P restricted to M is the cell complex obtained by intersecting each face of the diagram with M [20].

That is, the restricted Voronoi diagram of P on M is given by the intersections of M with the Voronoi regions of the Euclidean Voronoi diagram of P. The restricted Delaunay triangulation then follows as the set of Delaunay facets dual to the Voronoi edges

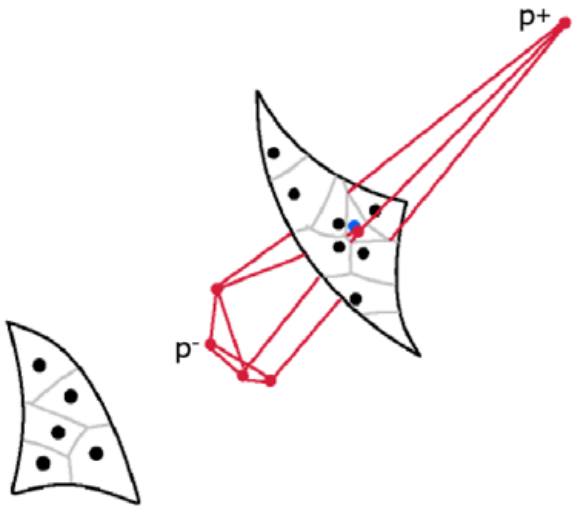

Figure B.2: In the 3D case, only a subset of the (red) vertices of a (blue) sample's (red) Voronoi region is located close to the medial axis. In this example, one Voronoi vertex lies close to the surface, equidistant from the four samples near the centre. The other vertices are located near the medial axis, specifically near the centre of curvature on one side and halfway to an opposite patch of the surface on the other side. The two vertices p^+, p^- (poles) farthest from the sample on either side of the surface are chosen as estimates of the medial axis of the sample. This figure is adapted from [8].

intersecting M:

Definition 5 The Delaunay triangulation of P restricted to M is the subcomplex of the triangulation consisting of those facets whose dual Voronoi edges intersect M [20].

This completes the presentation of various concepts from computational geometry referred to repeatedly in this thesis. For more details, see the references provided.

Appendix C

Data sources

The *dinosaur*, *Santa* and *sphere* surfel sets were obtained from the Pointshop3D [178] website hosted by the point-based graphics group at the ETH Zürich, Switzerland.

The *Isis*, *Venus*, *screwdriver* and *rocker arm* data sets were taken from the Cyberware Inc. website.

The *Bunny* and *Lucy* data sets are available from the Stanford 3D Scanning Repository at Stanford University, USA.

The *Michelangelo David*, *Day*, *Dawn*, *Youthful* and *St. Matthew* point sets were obtained from the Digital Michelangelo Project website. The permission to access these data sets granted by Marc Levoy is gratefully acknowledged.

The *Lena* and *Monarch* test images were taken from the Waterloo BragZone "Color Set" repertoire. The *Mandrill* image was obtained from the University of Southern California, USA, image database. The *Mona Lisa* image is available from The Art Platform [157].

Bibliography

[1] A. Adamson and M. Alexa, *Approximating and intersecting surfaces from points*, in Proc. SIGGRAPH '03, 2003, pp. 230–239.

[2] L. Adi and R. Kimmel, *Interactive edge integration on painted surfaces*, tech. report, Technion, Israel Institute of Technology, Tel-Aviv, Israel, 2001. http://www.cs.technion.ac.il/~liav/.

[3] M. Alexa and A. Adamson, *On normals and projection operators for surfaces defined by point sets*, in Proc. Eurographics Symp. on Point-Based Graph., 2004, pp. 149–156.

[4] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, *Point set surfaces*, in Proc. 12th IEEE Visual. Conf., 2001, pp. 21–28.

[5] ——, *Computing and rendering point set surfaces*, IEEE Trans. on Visual. and Comp. Graph., 9 (2003), pp. 3–15.

[6] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun, *Anisotropic polygonal remeshing*, in Proc. SIGGRAPH '03, 2003, pp. 485–493.

[7] P. Alliez, M. Meyer, and M. Desbrun, *Interactive geometry remeshing*, in Proc. SIGGRAPH '02, 2002, pp. 347–354.

[8] N. Amenta, M. Bern, and M. Kamvysselis, *A new voronoi-based surface reconstruction algorithm*, in Proc. SIGGRAPH '98, 1998, pp. 415–422.

[9] N. Amenta, S. Choi, T. K. Dey, and N. Leekha, *A simple algorithm for homeomorphic surface reconstruction*, in Proc. 16th ACM Symp. on Computat. Geom., 2000, pp. 213–222.

[10] N. Amenta, S. Choi, and R. Kolluri, *The power crust*, in Proc. 6th ACM Symp. on Solid Modeling, 2001, pp. 249–260.

[11] N. Amenta and Y. J. Kil, *Defining point set surfaces*, in Proc. SIGGRAPH '04, 2004, pp. 264–270.

[12] M. Andersson, J. Giesen, M. Pauly, and B. Speckmann, *Bounds on the k-neighborhood for locally uniformly sampled surfaces*, in Proc. Eurographics Symp. on Point-Based Graph., 2004, pp. 167–171.

[13] T. J. Barth and J. A. Sethian, *Numerical schemes for the hamilton-jacobi and level set equations on triangulated domains*, Journal of Computat. Phys., 145 (1998), pp. 1–40.

[14] M. Belkin and P. Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, Tech. Report CS TR-2002-01, University of Chicago, USA, 2002.

[15] F. Bernadini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, *The ball-pivoting algorithm for surface reconstruction*, IEEE Trans. on Visual. and Comp. Graph., 5 (1999), pp. 349–359.

[16] H. Biermann and D. Zorin, *Subdivide 2.0.* http://mrl.nyu.edu/ biermann/subdivision/.

[17] *Blitz++ Numerical Library, Vers. 0.7.* http://www.oonumerics.org/blitz/.

[18] J.-D. Boissonnat, *Geometric structures for three-dimensional shape representation*, ACM Trans. on Graph., 3 (1984), pp. 266–286.

[19] J.-D. Boissonnat and F. Cazals, *Smooth surface reconstruction via natural neighbour interpolation of distance functions*, in Proc. 16th ACM Symp. on Computat. Geom., 2000, pp. 223–232.

[20] ——, *Natural neighbor coordinates of points on a surface*, Computat. Geom.: Theory and Appl., 19 (2001), pp. 155–173.

[21] J.-D. Boissonnat and S. Oudot, *Provably good surface sampling and approximation*, in Proc. 1st Eurographics Symp. on Geom. Proc., 2003, pp. 246–265.

[22] M. Botsch, A. Wiratanaya, and L. Kobbelt, *Efficient high quality rendering of point sampled geometry*, in Proc. 13th Eurographics Workshop on Rendering, 2002, pp. 53–64.

[23] J. L. Bougrenet and J. F. Cavassilas, *Image coding using an adaptive sampling technique*, Signal Processing: Image Communication, 1 (1989), pp. 75–80.

[24] S. R. Buss and J. P. Fillmore, *Spherical averages and applications to spherical splines and interpolation*, ACM Trans. on Graph., 20 (2001), pp. 95–126.

[25] J. C. Carr, K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, *Reconstruction and representation of 3d objects with radial basis functions*, in Proc. SIGGRAPH '01, 2001, pp. 67–76.

[26] E. Catmull and J. Clark, *Recursively generated b-spline surfaces on arbitrary topological meshes*, Computer Aided Design, 10 (1978), pp. 350–355.

[27] *CGAL-2.4 - Computational Geometry Algorithms Library*. http://www.cgal.org.

[28] I. Chavel, *Riemannian geometry: a modern introduction*, Cambridge University Press, Cambridge, UK, 1997.

[29] L. P. Chew, *Guaranteed-quality mesh generation for curved surfaces*, in Proc. 9th Symp. on Computat. Geom., 1993, pp. 274–280.

[30] L. P. Chew and R. L. Drysdale, *Voronoi diagrams based on convex distance functions*, in Proc. 1st ACM Symp. on Comput. Geom., 1985, pp. 235–244.

[31] P. Cignoni, P. Montani, and R. Scopigno, *A comparison of mesh simplification algorithms*, Computers & Graphics, 22 (1998), pp. 37–54.

[32] P. Cignoni, C. Rocchini, and R. Scopigno, *Metro: measuring error on simplified surfaces*, Computer Graphics Forum, 17 (1998), pp. 167–174.

[33] L. D. Cohen, *Multiple contour finding and perceptual grouping using minimal paths*, Journal of Math. Imaging and Vision, 14 (2001), pp. 225–236.

[34] R. Cook, *Stochastic sampling in computer graphics*, ACM Trans. on Graph., 5 (1986), pp. 51–72.

[35] J. Cortés and F. Bullo, *Coordination and geometric optimization via distributed dynamical systems*, SIAM Journal on Control and Optimization, submitted, (2003).

[36] D. Cross, *Fast Fourier Transforms*. http://groovit.disjunkt.com/analog/time-domain/fft.html.

[37] O. Cuisenaire, *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*, PhD thesis, Université catholique de Louvain, Laboratoire de Telecommunications et Teledetection, Belgium, 1999.

[38] B. Curless and M. Levoy, *A volumetric method for building complex models from range images*, in Proc. SIGGRAPH '96, 1996, pp. 303–312.

[39] L. Demaret, N. Dyn, M. S. Floater, and A. Iske, *Adaptive thinning for terrain modeling and image compression*, in N. A. Dodgson, M. S. Floater and M. A. Sabin (eds.), Advances in Multiresolution for Geometric Modelling, Springer-Verlag, 2004, pp. 319–338.

[40] T. Deschamps and L. D. Cohen, *Fast extraction of minimal paths in 3d images and applications to virtual endoscopy*, Medical Image Analysis, 5 (2001), pp. 281–299.

[41] T. K. Dey, J. Giesen, and J. Hudson, *Decimating samples for mesh simplification*, in Proc. 13th Canadian Conf. on Comput. Geom., 2001, pp. 85–88.

[42] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[43] M. A. Z. Dippé and E. H. Wold, *Antialiasing through stochastic sampling*, in Proc. SIGGRAPH '85, 1985, pp. 69–78.

[44] N. Dyn and D. Levin, *Subdivision schemes in geometric modelling*, Acta Numerica, 12 (2002), pp. 73–144.

[45] H. Edelsbrunner and E. P. Mücke, *Three-dimensional alpha shapes*, ACM Trans. on Graph., 13 (1994), pp. 43–72.

[46] Y. Eldar, *Irregular image sampling using the voronoi diagram*, master's thesis, Technion, Israel Institute of Technology, Tel-Aviv, Israel, 1992.

[47] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi, *The farthest point strategy for progressive image sampling*, IEEE Trans. on Image Proc., 6 (1997), pp. 1305–1315.

[48] L. H. Figueiredo, J. de Miranda Gomes, D. Terzopoulos, and L. Velho, *Physically-based methods for polygonization of implicit surfaces*, Proc. Graphics Interface, (1992), pp. 250–257.

[49] S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva, *Progressive point set surfaces*, ACM Trans. on Graph., 22 (2003), pp. 997–1011.

[50] P. T. Fletcher, C. Lu, and S. Joshi, *Statistics of shape via principal geodesic analysis on lie groups*, in Proc. IEEE Conf. on Comp. Vision and Pattern Rec., 2003, pp. 95–101.

[51] M. S. Floater and K. Hormann, *Surface parameterization: a tutorial and survey*, in N. A. Dodgson, M. S. Floater and M. A. Sabin (eds.), Advances in Multiresolution for Geometric Modelling, Springer-Verlag, 2005, pp. 157–186.

[52] M. Fowler and K. Scott, *UML Distilled, 2nd ed.*, Addison-Wesley, Reading, USA, 2000.

[53] M. Garland and P. S. Heckbert, *Surface simplification using quadric error metrics*, in Proc. SIGGRAPH '97, 1997, pp. 209–216.

[54] J. Giesen and U. Wagner, *Shape dimension and intrinsic metric from samples of manifolds with high co-dimension*, in Proc. 19th ACM Symp. on Computat. Geom., 2003, pp. 329–337.

[55] A. S. Glassner, *An introduction to ray tracing*, Morgan Kaufmann, 1989.

[56] ——, *Principles of digital image synthesis*, Morgan Kaufmann, 1995.

[57] T. F. Gonzalez, *Clustering to minimize the maximum intercluster distance*, Theor. Comp. Sc., 38 (1985), pp. 293–306.

[58] C. Gotsman, S. Gumhold, and L. Kobbelt, *Simplification and compression of 3d meshes*, in A. Iske, E. Quak and M. S. Floater (eds.), Tutorials on Multiresolution in Geometric Modelling, Springer-Verlag, 2002, pp. 319–362.

[59] J. P. Grossman and W. J. Dally, *Point sample rendering*, in Proc. 9th Eurographics Workshop on Rendering, 1998, pp. 181–192.

[60] M. Grundland, C. Gibbs, and N. A. Dodgson, *Stylized rendering for multiresolution image representation*, in Proc. Human Vision and Electronic Imaging X Conf., to appear, 2005.

[61] S.-S. Guan and M. R. Luo, *A colour-difference formula for assessing large colour differences*, Color Res. and Appl., 24 (1999), pp. 344–355.

[62] G. Guennebaud, L. Barthe, and M. Paulin, *Dynamic surfel set refinement for high quality rendering*, Computers & Graphics, 28 (2004), pp. 827–838.

[63] ——, *Real-time point cloud refinement*, in Proc. Eurographics Symp. on Point-Based Graph., 2004, pp. 41–49.

[64] S. Gumhold, X. Wang, and R. McLeod, *Feature extraction from point clouds*, in Proc. 10th Int. Meshing Roundtable, 2001, pp. 293–305.

[65] I. Guskov and Z. Wood, *Topological noise removal*, in Proc. Graphics Interface, 2001, pp. 19–26.

[66] G. Guy and G. Medioni, *Inference of surfaces, 3d curves and junctions from sparse, noisy 3d data*, IEEE Trans. Pattern Anal. and Mach. Intell., 19 (1997), pp. 1265–1277.

[67] M. Hall and J. Warren, *Adaptive polygonization of implicitly defined surfaces*, IEEE Comp. Graph. and Appl., 10 (1990), pp. 33–42.

[68] J. H. Halton, *On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals*, Num. Math., 2 (1960), pp. 84–90.

[69] A. B. Hamza and H. Krim, *Geodesic object representation and recognition*, Lecture Notes in Computer Science, 2886 (2003), pp. 378–387.

[70] J. Helmsen, E. G. Puckett, P. Collela, and M. Dorr, *Two new methods for simulating photolithography development in 3d*, in Proc. SPIE Microlithography IX, 1996, pp. 253–261.

[71] H. Hoppe, *Progressive meshes*, Proc. SIGGRAPH '96, (1996), pp. 99–108.

[72] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, *Surface reconstruction from unorganized points*, in Proc. SIGGRAPH '92, 1992, pp. 19–26.

[73] ——, *Mesh optimization*, in Proc. SIGGRAPH '93, 1993, pp. 19–26.

[74] F. John, *Partial Differential Equations, 4th ed.*, Springer-Verlag, 1982.

[75] J. Jost, *Equilibrium maps between metric spaces*, Calculus of Variations and Partial Differential Equations, 2 (1994), pp. 173–204.

[76] A. Kalaiah and A. Varshney, *Differential point rendering*, in Proc. 12th Eurographics Workshop on Rendering, 2001, pp. 139–150.

[77] ——, *Modeling and rendering of points with local geometry*, IEEE Trans. on Visual. and Comp. Graph., 9 (2003), pp. 30–42.

[78] ——, *Statistical point geometry*, in Proc. 1st Eurographics Symp. on Geom. Proc., 2003, pp. 107–115.

[79] H. Karcher, *Riemannian center of mass and mollifier smoothing*, Comm. on Pure and Appl. Math., 30 (1977), pp. 509–541.

[80] A. Keller, *Strictly deterministic sampling methods in computer graphics*, SIGGRAPH '03 course #44, Monte Carlo Ray Tracing, (2003).

[81] W. S. Kendall, *Probability, convexity and harmonic maps with small image i: Uniqueness and fine existence*, Proc. London Math. Soc., 61 (1990), pp. 371–406.

[82] L. Kettner, *Using generic programming for designing a data structure for polyhedral surfaces*, Computat. Geom.: Theory and Appl., 13 (1999), pp. 65–90.

[83] R. Kimmel, N. Kiryati, and A. M. Bruckstein, *Sub-pixel distance maps and weighted distance transforms*, Journal of Math. Imaging and Vision, 6 (1996), pp. 223–233.

[84] R. Kimmel and J. A. Sethian, *Computing geodesic paths on manifolds*, Proc. Nat. Acad. of Sciences, 95 (1998), pp. 8431–8435.

[85] ——, *Fast voronoi diagrams and offsets on triangulated surfaces*, in A. Cohen, J. L. Merrien and L. L. Schumaker (eds.), Proc. Curves and Surfaces, St. Malo, France, Nashboro Press, 1999.

[86] L. P. Kobbelt, S. Bischoff, M. Botsch, K. Kaehler, C. Rössl, R. Schneider, and J. Vorsatz, *Geometric modelling based on polygonal meshes*, Eurographics '00 tutorial, (2000).

[87] L. P. Kobbelt and M. Botsch, *A survey of point-based techniques in computer graphics*, Computers & Graphics, to appear, (2004).

[88] R. Kunze, F.-E. Wolter, and T. Rausch, *Geodesic voronoi diagrams on parametric surfaces*, in Proc. Graphics Interface, 1997, pp. 230–238.

[89] J. M. Lane and R. F. Riesenfeld, *A theoretical development for the computer generation of piecewise polynomial surfaces*, IEEE Trans. Pattern Anal. and Mach. Intell., 2 (1980), pp. 35–46.

[90] G. Leibon and D. Letscher, *Delaunay triangulations and voronoi diagrams for riemannian manifolds*, in Proc. 16th ACM Symp. on Computat. Geom., 2000, pp. 341–349.

[91] R. J. LeVeque, *Finite difference methods for differential equations*, tech. report, University of Washington, Dept. of Appl. Math., Washington, USA, 1998. Draft lecture notes.

[92] D. Levin, *The approximation power of moving least-squares*, Math. of Comp., 67 (1998), pp. 1517–1531.

[93] ——, *Mesh-independent surface interpolation*, in G. Brunett (ed.), Geometric Modeling for Scientific Visualization, Springer-Verlag, 2003, pp. 37–49.

[94] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, *The digital michelangelo project: 3d scanning of large statues*, in Proc. SIGGRAPH '00, 2000, pp. 131–144.

[95] M. Levoy and T. Whitted, *The use of points as display primitive*, Tech. Report 85022, UNC-Chapel Hill Computer Science Dept., USA, 1985. http://graphics.stanford.edu/papers/points/point-with-scanned-figs.pdf.

[96] P. Lindstrom, *Out-of-core simplification of large polygonal models*, in Proc. SIGGRAPH '00, 2000, pp. 259–262.

[97] L. Linsen, *Point cloud representation*, Tech. Report 2001-3, Computer Science Dept., Universität Karlsruhe, Germany, 2001. http://www.ubka.uni-karlsruhe.de/vvv/ira/2001/3/3.pdf.

[98] C. Loop, *Smooth surface subdivision based on triangles*, master's thesis, Dept. of Math., University of Utah, USA, 1987.

[99] W. E. Lorensen and H. E. Cline, *Marching cubes: A high resolution 3d surface construction algorithm*, in Proc. SIGGRAPH '87, 1987, pp. 163–169.

[100] K.-L. Low and T.-S. Tan, *Model simplification using vertex-clustering*, in Proc. Symp. Interactive 3D Graph., 1997, pp. 75–81.

[101] D. K. McAllister, L. F. Nyland, V. Popescu, A. Lastra, and C. McCue, *Real-time rendering of real-world environments*, in Proc. 10th Eurographics Workshop on Rendering, 1999, pp. 145–160.

[102] K. McLaren, *The development of the cie 1976 (l*a*b*) uniform colour-space and colour-difference formula*, Journal of the Soc. of Dyers and Colourists, 92 (1976), pp. 338–341.

[103] F. Mémoli and G. Sapiro, *Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces*, Journal of Comput. Phys., 173 (2001), pp. 764–795.

[104] ——, *Distance functions and geodesics on point clouds*, Tech. Report 1902, Institute for Mathematics and its Applications, University of Minnesota, USA, 2003. http://www.ima.umn.edu/preprints/dec2002/1902.pdf, to appear in SIAM Journal of Appl. Math.

[105] ——, *Comparing point clouds*, in Proc. 2nd Eurographics Symp. on Geom. Proc., 2004.

[106] ——, *A theoretical and computational framework for isometry invariant recognition of point cloud data*, Tech. Report 1800, Institute for Mathematics and its Applications, University of Minnesota, USA, 2004. http://www.ima.umn.edu/preprints/jun2004/1980.pdf.

[107] D. P. Mitchell, *Spectrally optimal sampling for distribution ray tracing*, in Proc. SIGGRAPH '91, 1991, pp. 157–164.

[108] J. S. B. Mitchell, *Geometric shortest paths and network optimization*, in J.-R. Sack and J. Urrutia (eds.), Handbook of Computat. Geom., Elsevier, 2000, pp. 633–701.

[109] N. J. Mitra and A. Nguyen, *Estimating surface normals in noisy point cloud data*, in Proc. 19th ACM Symp. on Comput. Geom., 2003, pp. 322–328.

[110] C. Moenning and N. A. Dodgson, *Fast marching farthest point sampling*, Tech. Report 562, Computer Laboratory, University of Cambridge, UK, 2003. http://www.cl.cam.ac.uk/~cm230/FastFPS.pdf.

[111] ——, *Fast marching farthest point sampling*, in Proc. Eurographics '03, 2003. Poster paper.

[112] ——, *A new point cloud simplification algorithm*, in Proc. 3rd Int. Conf. on Visual., Imaging and Image Proc., 2003, pp. 1027–1033.

[113] ——, *Intrinsic point cloud simplification*, in Proc. 14th GraphiCon '04, 2004.

[114] C. Moenning, F. Mémoli, G. Sapiro, N. Dyn, and N. A. Dodgson, *Meshless geometric subdivision*, Tech. Report 1977, Institute for Mathematics and its Applications, University of Minnesota, USA, 2004. http://www.ima.umn.edu/preprints/apr2004/1977.pdf.

[115] ——, *Meshless geometric subdivision*, Graphical Models, submitted, (2005).

[116] D. Nehab and P. Shilane, *Stratified point sampling of 3d models*, in Proc. Eurographics Symp. on Point-Based Graph., 2004, pp. 49–56.

[117] *Newmat C++ matrix library, Vers. 11.* http://www.robertnz.net/.

[118] M. Novotni and R. Klein, *Computing geodesic distances on triangular meshes*, in Proc. 10th Int. Conf. in Central Europe on Comp. Graph., Visual. and Comp. Vision (WSCG), 2002, pp. 341–347.

[119] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel, *Multi-level partition of unity implicits*, in Proc. SIGGRAPH '03, 2001, pp. 463–470.

[120] A. Okabe, B. Boots, and K. Sugihara, *Spatial Tessellations - Concepts and Applications of Voronoi Diagrams, 2nd ed.*, John Wiley & Sons, Chicester, UK, 2000.

[121] P. Oswald and P. Schröder, *Composite primal/dual sqrt(3)-subdivision schemes*, Comp. Aided Geom. Design, 20 (2003), pp. 135–164.

[122] M. Pauly, *Point Primitives for Interactive Modeling and Processing of 3D Geometry*, PhD thesis, Computer Science Department, ETH Zürich, Switzerland, 2004.

[123] M. Pauly and M. Gross, *Spectral processing of point-sampled geometry*, in Proc. SIGGRAPH '01, 2001, pp. 379–386.

[124] M. Pauly, M. Gross, and L. P. Kobbelt, *Efficient simplification of point-sampled surfaces*, in Proc. 13th IEEE Visual. Conf., 2002, pp. 163–170.

[125] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross, *Shape modeling with point-sampled geometry*, in Proc. SIGGRAPH '03, 2003, pp. 641–650.

[126] M. Pauly, L. Kobbelt, and M. Gross, *Multiresolution modeling of point-sampled geometry*, Tech. Report 378, Computer Science Department, ETH Zürich, Switzerland, 2002. http://graphics.stanford.edu/~mapauly/Pdfs/MultiresModeling.pdf.

[127] G. Peyré and L. Cohen, *Geodesic re-meshing using front propagation*, in Proc. 2nd IEEE Workshop on Variational, Geometric and Level Set Methods in Comp. Vis. (VLSM), 2003.

[128] ——, *Surface segmentation using geodesic centroidal tesselation*, in Proc. 2nd Symp. 3D Data Proc., Visual. and Trans. (3DPVT), 2004, pp. 995–1002.

[129] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, *Surfels: Surface elements as rendering primitives*, in Proc. SIGGRAPH '00, 2000, pp. 335–342.

[130] *PointsToPolys software.* http://www.paraform.com/ppdl.

[131] W. H. Press, S. A. Teukolksy, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C, 2nd ed.*, Cambridge University Press, Cambridge, UK, 1992.

[132] *RapidForm 2004.* http://www.rapidform.com.

[133] M. Reimers, *Computing distances on 3d triangular meshes*, submitted, (2004).

[134] H. L. Resnikoff, *The duality between noise and aliasing and human image understanding*, SPIE Image Understanding and Man-Machine Interface, 758 (1987), pp. 31–38.

[135] J. Rossignac and P. Borrel, *Multi-resolution 3d approximations for rendering complex scenes*, in Proc. Conf. on Geom. Model. in Comp. Graph., 1993, pp. 455–465.

[136] E. Rouy and A. Tourin, *A viscosity solutions approach to shape-from-shading*, SIAM Journal of Num. Anal., 29 (1992), pp. 867–884.

[137] M. Roy, F. Nicolier, S. Foufou, F. Truchetet, A. Koschan, and M. Abidi, *Assessment of mesh simplification algorithm quality*, in Proc. SPIE Electronic Imaging, 2002, pp. 128–137.

[138] J. Ruppert, *A delaunay refinement algorithm for quality 2-dimensional mesh generation*, Journal of Algorithms, 18 (1995), pp. 548–585.

[139] S. Rusinkiewicz and M. Levoy, *Qsplat: a multiresolution point rendering system for large meshes*, in Proc. SIGGRAPH '00, 2000, pp. 343–352.

[140] R. Schaback, *Creating surfaces from scattered data using radial basis functions*, in M. Daehlen, T. Lyche and L. Schumaker (eds.), Mathematical Models in Computer Aided Geometric Design III, USA, 1995, Vanderbilt University Press, pp. 477–496.

[141] R. Scopigno, C. Andujar, M. Gösele, and H. Lensch, *3d data acquisition*, Eurographics '02 tutorial, (2002).

[142] R. Sedgewick, *Algorithms in C, 3rd ed.*, Addison-Wesley, Reading, USA, 1998.

[143] ——, *Algorithms in C++ - Part 5: Graph Algorithms, 3rd ed.*, Addison-Wesley, Reading, USA, 2001.

[144] J. A. Sethian, *Theory, algorithms, and applications of level set methods for propagating interfaces*, Acta Numerica, 5 (1996), pp. 309–395.

[145] ——, *Level Set Methods and Fast Marching Methods, 2nd ed.*, Cambridge University Press, Cambridge, UK, 1999.

[146] J. A. Sethian and A. Vladimirsky, *Fast methods for the eikonal and related hamilton-jacobi equations on unstructured meshes*, Proc. Nat. Acad. of Sciences, 97 (2000), pp. 5699–5703.

[147] ——, *Ordered upwind methods for static hamilton-jacobi equations*, Proc. Nat. Acad. of Sciences, 98 (2001), pp. 11069–11074.

[148] R. Shahidi, C. Moloney, and G. Ramponi, *Design of farthest-point masks for image halftoning*, EURASIP Journal on Appl. Signal Proc., 12 (2004), pp. 1886–1898.

[149] A. Sharf, M. Alexa, and D. Cohen-Or, *Context-based surface completion*, ACM Trans. on Graph., 23 (2004), pp. 878–887.

[150] P. Shirley, *Discrepancy as a quality measure for sample distributions*, in Proc. Eurographics '91, 1991, pp. 183–193.

[151] R. Sibson, *A vector identity for the dirichlet tessellation*, in Math. Proc. of the Cambridge Philosoph. Soc., 1980, pp. 151–155.

[152] O. Sifri, A. Sheffer, and C. Gotsman, *Geodesic-based surface remeshing*, in Proc. 12th Int. Meshing Roundtable, 2003, pp. 189–199.

[153] M. Stamminger and G. Drettakis, *Interactive sampling and rendering for complex and procedural geometry*, in Proc. 12th Eurographics Workshop on Rendering, 2001, pp. 151–162.

[154] G. Taubin, *Estimating the tensor of curvature of a surface from a polyhedral approximation*, in Proc. 5th Int. Conf. on Comp. Vis. (ICCV), 1995, pp. 902–909.

[155] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*, Kluwer, Boston, USA, 2002.

[156] J. B. Tenenbaum, V. de Silva, and J. C. Langford, *A global geometric framework for nonlinear dimensionality reduction*, Science, 290 (2000), pp. 2319–2323.

[157] *The Art Platform*. http://www.art-platform.com.

[158] J. N. Tsitsiklis, *Efficient algorithms for globally optimal trajectories*, IEEE Trans. on Automatic Control, 40 (1995), pp. 1528–1538.

[159] G. Turk, *Re-tiling polygonal surfaces*, in Proc. SIGGRAPH '92, 1992, pp. 55–64.

[160] R. Ulichney, *Digital Halftoning*, MIT Press, 1987.

[161] ——, *Dithering with blue noise*, Proc. of the IEEE, 76 (1988), pp. 56–79.

[162] L. Velho, J. Gomes, and L. H. de Figueiredo, *Implicit Objects in Computer Graphics*, Springer-Verlag, New York, USA, 2002.

[163] J. Vorsatz, C. Rössl, L. P. Kobbelt, and H.-P. Seidel, *Feature sensitive remeshing*, in Proc. Eurographics '01, 2001, pp. 393–401.

[164] J. Wallner and N. Dyn, *Smoothness of subdivision schemes by proximity*, Tech. Report 112, Tech. Univ. Wien, Austria, 2003.

[165] ——, *Smoothness of subdivision schemes in manifolds*, Tech. Report 125, Tech. Univ. Wien, Austria, 2004. http://www.geometrie.tuwien.ac.at/wallner/sbd2.pdf.

[166] ——, *Convergence and c1 analysis of subdivision schemes on manifolds by proximity*, Comp. Aided Geom. Design, to appear, 22 (2005). http://www.geometrie.tuwien.ac.at/wallner/sbd1.pdf.

[167] M. Wand, M. Fischer, and F. M. auf der Heide, *The randomized z-buffer algorithm: Interactive rendering of highly complex scenes*, in Proc. SIGGRAPH '01, 2001, pp. 361–370.

[168] A. Watt and M. Watt, *Advanced Animation and Rendering Techniques*, Addison-Wesley, 1992.

[169] T. Weyrich, M. Pauly, R. Keiser, S. Heinzle, S. Scandella, and M. Gross, *Post-processing of scanned 3d surface data*, in Proc. Eurographics Symp. on Point-Based Graph., 2004, pp. 85–94.

[170] *Wild Magic 2.4*. http://www.magic-software.com/SourceCode.html.

[171] A. P. Witkin and P. S. Heckbert, *Using particles to sample and control implicit surfaces*, in Proc. SIGGRAPH '94, 1994, pp. 269–277.

[172] T.-T. Wong, W.-S. Luk, and P.-A. Heng, *Sampling with hammersley and halton points*, Journal of Graph. Tools, 2 (1997), pp. 9–24.

[173] Z. Wood, H. Hoppe, M. Desbrun, and P. Schröder, *Removing excess topology from isosurfaces*, ACM Trans. on Graph., 23 (2004), pp. 190–208.

[174] J. Wu and L. Kobbelt, *Optimized sub-sampling of point sets for surface splatting*, in Proc. Eurographics '04, 2004, pp. 643–652.

[175] H.-K. Zhao, S. Osher, and R. Fedkiw, *Fast surface reconstruction using the level set method*, in Proc. 1st IEEE Workshop on Variational, Geometric and Level Set Methods in Comp. Vis. (VLSM), 2001.

[176] G. Zigelman, R. Kimmel, and N. Kiryati, *Texture mapping using surface flattening via multidimensional scaling*, IEEE Trans. on Visual. and Comp. Graph., 8 (2002), pp. 198–207.

[177] D. Zorin and P. Schröder, *Subdivision for modeling and animation*, SIGGRAPH '00 course notes, (2000).

[178] M. Zwicker, M. Pauly, M. Knoll, and M. Gross, *Pointshop3d: An interactive system for point-based surface editing*, in Proc. SIGGRAPH '02, 2002, pp. 322–329.

[179] M. Zwicker, H. Pfister, J. van Baar, and M. Gross, *Surface splatting*, in Proc. SIGGRAPH '01, 2001, pp. 371–378.

[180] ——, *Ewa splatting*, IEEE Trans. on Visual. and Comp. Graph., 8 (2002), pp. 223–238.