SURFACE RECONSTRUCTION METHODS FOR SEAFLOOR MODELLING

Ricard Campos Dausà

Dipòsit legal: Gi. 1110-2014
http://hdl.handle.net/10803/145380

Universitat de Girona

PhD Thesis

Surface Reconstruction Methods for Seafloor Modelling

Ricard Campos Dausà

2014

Universitat de Girona

PhD Thesis

Surface Reconstruction Methods for Seafloor Modelling

Ricard Campos Dausà

2014

Doctoral Programme in Technology

Supervised by:

Rafael Garcia

Work submitted to the University of Girona in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Universitat de Girona

Dr. Rafael Garcia, from Universitat de Girona,

DECLARES

That the work entitled *Surface Reconstruction Methods for Seafloor Modelling* presented by *Ricard Campos Dausà* to obtain the degree in Doctor of Philosophy has been developed under my supervision and complies with the requirements needed to obtain the International Mention.

Therefore, in order to certify the aforesaid statement, I sign this document.

Girona, March 2014.

A l'Aïda i els meus pares...

Acknowledgements

I would like to start by expressing my gratitude to my supervisor, Dr. Rafael Garcia, for his wise advises and for encouraging me through these years. I am especially thankful for the patience you had with all the changes that this thesis has suffered...

Also, to my colleagues at the Underwater Vision Laboratory who had always been there when I needed them. To the ones that are currently on the group, Tudor, Nuno, Quintana, Ricard P., Mojdeh, Konstantin, Shihav, László, and also to the ones that left, Arman, Olivier, Pio and Ramon (and many others I probably forget). Also to the colleagues at CIRS, with whom I shared the pressure of many field experiments. Of course, I would like to extend my gratitude to VICOROB as a whole, and particularly to Xavi Lladó and Jordi Freixenet for introducing me into the group during the development of my final degree project.

Obviously, I would like to especially thank Gerard, Pablo, Mariano, Albert Pla and Albert Gubern, who started this journey with me from the very beginning, and with whom I shared many unforgettable moments. I also thank the ones who appeared a bit after that: Masi, a hard worker, and Enric, the only one concerned with providing me data to use in this thesis!. Moreover, I thank as well the last incorporations, Eloy and Pepe, for the fruitful discussions at coffee time.

Likewise, I am really grateful to Mariette Yvinec and Pierre Alliez, and in general to all the people at Geometrica (INRIA), for their hospitality during my stay there. All the ideas shared and the experience gained there with you have greatly contributed to the development of this thesis.

Of course, I also want to thank both the reviewers and the jury of the thesis, for taking some of your time in evaluating my work.

Moreover, regarding part of the datasets used in this thesis, I wish to thank the Stanford 3D Scanning Repository (Stanford University Computer Graphics Laboratory), the AIM@SHAPE consortium, H. Hoppe, Y. Furukawa, Y. Nagai and C. Strecha for making their data publicly available, and also R. Keriven and J.-P. Pons for providing some models used in this thesis. I would also like to express my gratitude to N. Amenta, F. Calakly, B. Curless, T. K. Dey, G. Guennebaud, M. Kazhdan, V. Lempitsky, J. Manson, Y. Nagai, Y.

Ohtake and R. R. Paulsen (and their collaborators), along with the teams of the CGAL libraries and the MeshLab software for making their code and/or executables available to the community. Additionally, I thank the agencies that partially funded this work: the Spanish Ministry of Science and Innovation under grant CTM2010-15216 (MuMap) and the EU projects FP7-ICT-2009-248497 (Trident), FP7-ICT-2011-7-288704 (Morph) and FP7-INF-2012-312762 (Eurofleets2).

Finalment, m'agradaria acabar aquests agraïments donant les gràcies a la meva família. Especialment, als meus pares Carmen i Antonio, per fer-me com sóc, i per haver-me donat sempre tot el suport i l'amor possible. També als meus germans, Xavi i Dani, per haver-me aguantat tots aquests anys. Així mateix, als meus avis que, encara que la majoria ja no hi siguin, sempre m'han animat a que seguís amb els meus estudis. I també a la meva *família política*, la Montse, la Luci i en Lluís, que sempre s'han preocupat perquè aquesta tesi arribés a bon port.

Reservo l'últim agraïment per l'Aïda, la persona més important de la meva vida. Gràcies per fer-me tant feliç, i per haver-me suportat i ajudat tant durant tots aquests anys.

Publications

The present thesis has led to the publication of the following technical articles:

- R. Campos, R. Garcia, P. Alliez, and M. Yvinec. “A surface reconstruction method for in-detail underwater 3D optical mapping”. *International Journal of Robotics Research*, Submitted, December 2013.
- R. Campos, R. Garcia, P. Alliez, and M. Yvinec. “Splat-based surface reconstruction from defect-laden point sets”. *Graphical Models*, 75(6): 346-361, 2013.
- E. Galceran, R. Campos, M. Carreras and P. Ridao. “3D Coverage Path Planning with Realtime Replanning for Inspection of Underwater Structures”. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014 (to appear).
- R. Campos, N. Gracias, R. Prados, and R. Garcia. “Merging bathymetric and optical cues for in-detail inspection of an underwater shipwreck”. *Instrumentation Viewpoint*, 2013.
- N. Gracias, P. Ridao, R. Garcia, J. Escartín, M. L’Hour, F. Cibecchini, R. Campos, M. Carreras, D. Ribas, N. Palomeras, L. Magi, A. Palomer, T. Nicosevici, R. Prados, R. Hegedus, L. Neumann, F. de Filippo, and A. Mallios. “Mapping the moon: Using a lightweight AUV to survey the site of the 17th century ship ‘La Lune”’. *MTS/IEEE OCEANS conference*, June 2013.
- P. Ridao, D. Ribas, N. Palomeras, M. Carreras, A. Mallios, N. Hurtós, N. Gracias, Ll. Magí, R. Garcia, R. Campos, R. Prados, and J. Escartín. “Operational validation of Girona500 AUV”. *Instrumentation Viewpoint*, 2013.
- M. Prats, D. Ribas, N. Palomeras, J. C. García, V. Nannen, S. Wirth, J. J. Fernández, J. P. Beltrán, R. Campos, P. Ridao, P. J. Sanz, G. Oliver, M. Carreras, N. Gracias, R. Marín, and A. Ortiz, “Reconfigurable AUV for intervention missions: a case study on underwater object recovery”. *Intelligent Service Robotics*, 5(1):19-31, 2012.

- P. Nomikou, J. Escartín, P. Ridao, D. Sakellariou, R. Camilli, V. Ballu, M. Moreira, C. Mével, A. Mallios, C. Deplus, M. Andreani, O. Pot, R. Garcia, L. Rouzie, T. Gabsi, R. Campos, N. Gracias, N. Hurtós, L. Magi, N. Palomeras, and D. Ribas. "Preliminary submarine monitoring of Santorini caldera: Hydrothermal activity, and seafloor deformation". *Volcanism of the Southern Aegean in the frame of the broader Mediterranean area Conference* (Poster), October 2012.

- J. Quintana, R. Campos, R. Garcia, J. Freixenet, N. Gracias, S. Puig, and J. Malvehy. "A novel acquisition device for total body photography". *World Congress of Dermoscopy* (Abstract), 2012.

- R. Campos, R. Garcia, and T. Nicosevici. "Surface reconstruction methods for the recovery of 3D models from underwater interest areas". in *IEEE OCEANS Conference*, Spain, pp. 1-10, 2011.

- J. Quintana, R. Campos, N. Gracias, J. Freixenet, and R. Garcia. "3D skin mapping for melanoma detection". *Medical Image Computing Conference (MICCAT 2011)*, 2011.

- S. Zandara, P. Ridao, D. Ribas, R. Campos, and A. Mallios. "Kornati bathymetry survey data-set for navigation and mapping". in *19th Mediterranean Conference on, Control Automation (MED)*, pp. 443-448, 2011.

- R. Garcia, R. Campos, and J. Escartín. "High-resolution 3D reconstruction of the seafloor for environmental monitoring and modelling". in *IROS 2011, Workshop on Robotics for Environmental Monitoring*, September 2011.

- J. Escartín, R. Garcia, R. Prados, R. Campos, T. Barreyre, and Bathyluck09 Science Party. "Image seafoor mosaics: Acquisition, processing, and role on deep-sea observatory planning and implementation". *Geophysical Research Abstracts*, vol. 12, 2010.

- R. Campos, J. Ferrer, M. Villanueva, L. Magí, and R. Garcia. "Trinocular system for 3D motion and dense structure estimation". *Instrumentation Viewpoint*, 2009.

List of Acronyms

AABB Axis-Aligned Bounding Boxes

APSS Algebraic Point Set Surfaces

AUV Autonomous Underwater Vehicle

BA Bundle Adjustment

BBD Bounding Box Diagonal

CSG Constructive Solid Geometry

CSRBF Compactly Supported Radial Basis Function

DLT Direct Linear Transformation

DM Discrete Membrane

DoG Difference of Gaussians

DWT Discrete Wavelet Transform

FEM Finite Element Method

FFT Fast Fourier Transform

GCS Growing Cell Structure

GPU Graphics Processing Unit

LBQ Local Bivariate Quadric

LKS Least K-th Squares

LMedS Least Median of Squares

LoG Laplacian of Gaussian

MA	Medial Axis
MAD	Median Absolute Deviation
MAP	Maximum a Posteriori
MLS	Moving Least Squares
MPU	Multilevel Partition of Unity
MRF	Markov Random Field
MSER	Maximally Stable Extremal Regions
MSSE	Modified Selective Statistical Estimator
MST	Minimum Spanning Tree
NCC	Normalized Cross-Correlation
NN	Neural Network
PCA	Principal Components Analysis
PDE	Partial Differential Equation
PDF	Probability Density Function
PMVS	Patch-based Multi-View Stereo
PSS	Point Set Surfaces
RANSAC	RANdom SAmple Consensus
RBF	Radial Basis Function
RDT	Restricted Delaunay Triangulation
RKHS	Reproducing Kernel Hilbert Space
ROV	Remotely Operated Vehicle
SDF	Signed Distance Function
SFM	Structure-from-Motion
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping

SOM	Self-Organizing Map
SoS	Simulation of Simplicity
SSD	Sum of Squared Differences
SURF	Speeded Up Robust Features
SVM	Support Vector Machines
SVR	Support Vector Regression
UDF	Unsigned Distance Function
VRIP	Volumetric Range Image Processing

List of Symbols

Sets

$LoS(p_i)$	Set of Lines-of-Sight associated with point p_i.
$K_k^Y(x)$	Set of k-nearest neighbors of x from the set Y. When Y not specified, we assume the input point set P. Also, we omit k if it is a parameter.
$ON_{c,d}(v_i)$	Neighboring octree cell of v_i assuming connectivity c and depth d.
P^X	Points from the set X. When the set is not specified, the input point set is assumed.
$Q(P)$	Set of poles (positive and negative) of $\text{Vor}(P)$.
S	Ideal (unknown) surface.
$\bar{S}$	Interpolated surface.
$\tilde{S}$	Approximated surface.
G	Voxel grid.
$GN_c(v_i)$	Neighboring voxel of v_i inside G assuming connectivity c.

Simplicial Complexes

$\text{CH}(P)$	Convex hull of the set P.
$\text{Del}(P)$	Delaunay triangulation of P.
$\text{GC}(P)$	Gabriel Complex of P.
$\text{MA}(P)$	Medial Axis of P.
$\text{MAT}(P)$	Medial Axis Transform of P.

MS(P)	Medial Scaffold of P.
Vor(P)	Voronoi diagram of P.

Primitives

$B(x)$	Enclosing ball of entity x.
$B(q_i)$	Polar ball of q_i.
e_i	Edge i.
$e_{i,j}$	Edge joining p_i and p_j.
n_i	Normal vector associated to entity i. When not stated otherwise, n_i refers to the normal at p_i. In other cases, i may refer to the normal of a plane or a triangle.
$NR(V_{p_i}, p_j)$	Natural region: portion of the Voronoi cell defined by p_i disappearing upon insertion of p_j.
o_i^d	Octree node i at depth d, with an associated scalar value.
${}^{R}p_i^X$	Single point i from set or entity X. If set is not specified, the input point set P is assumed. Also, it may have an associated reference frame R.
q_i^+	Positive pole i generated by Vp_i.
q_i^-	Negative pole i generated by Vp_i.
t_i	Triangle i.
tet_i	Tetrahedron i.
$\vec{v}$	Vector.
V_p	Voronoi cell of Vor(P) corresponding to point p.
v_i	Voxel i, with an associated scalar value.
$v_{i,j,k}$	Voxel with index (i, j, k), with an associated scalar value.

Graphs

$\mathsf{e}(\mathsf{v}_i, \mathsf{v}_j)$	Edge of a graph joining the nodes v_i and v_j.
$\mathsf{e}_{i,j}$	Edge of a graph joining the nodes v_i and v_j.

G	Generic graph.
$MST(P)$	Minimum Spanning Tree of P.
v_i	Vertex/node i of a graph.
$w(\mathsf{e}(\mathsf{v}_i, \mathsf{v}_j))$	Weight of the edge joining v_i and v_j in a graph.

Functions

$\langle x, y\rangle$	Inner/dot product between x and y.
$\chi(x)$	Indicator function of a volume.
$\phi_i(x)$	Radial Basis Function i.
$\phi_i(x, c)$	Radial Basis Function i applied at x with center c.
$\phi_{i,r}(x)$	Compactly supported RBF i, having a support radius r.
$arclength(p_i, p_j)$	Length of the arc defined by p_i, p_j.
$area(x)$	Area of, or associated to, x.
$bbox(X)$	Axis-aligned bounding box enclosing all elements in set X.
$center(x)$	Center of the enclosing ball of primitive x.
$centroid(X)$	Centroid of X.
$diam(x)$	Diameter of the enclosing ball of primitive x.
$dihedral(x, y)$	Dihedral angle between two planes x and y. They may correspond to triangles, i.e., $x = t_i$, $y = t_j$.
$dist(x, y)$	Euclidean distance between entity x and y.
$dist_d(x, y)$	Distance between entity x and y, following norm d.
$dist_{flow}(x, y$	Flow distance between entity x and y.
$dist_{pow}(x, y)$	Power distance between entity x and y.
$dist_w(x, y)$	Wasserstein distance between entity x and y.
$exp(x)$	Exponential function. Used in some cases instead of e^x for readability.
$I(x)$	Implicit function.

$iangle(B(x), B(y))$	Intersection angle between balls $B(x)$ and $B(y)$.
$lfs(p_i)$	Local Feature Size of point p_i.
$midpoint(e_i)$	Midpoint of an edge e_i.
$\min(x, y)$	Minimum value between x and y.
$\max(x, y)$	Maximum value between x and y.
$odepth(p_i)$	Depth of the node where p_i lays in the octree.
$rad(x)$	Radius of the enclosing ball of primitive x.
$rad(q_i)$	Radius of the polar ball corresponding to q_i.
$vol(x)$	Volume of entity x.
Other	
$E(x)$	Energy to minimize in a variational procedure.
$\mathcal{P}(x)$	Probability of x.

List of Figures

List of Tables

Contents

Resum

Els mapes del fons del mar són una important font d'informació per la comunitat científica, donat que la cartografia del fons marí és el punt de partida per l'exploració dels oceans. Els avenços en les metodologies d'escaneig, mitjançant tècniques tant acústiques com òptiques, permet que la construcció de mapes del fons marí es realitzi cada cop a més altes resolucions. Tot i això, totes aquestes tècniques mostregen la superfície de l'àrea d'interès en la forma d'un núvol de punts. En el cas de les àrees que presenten un gran relleu 3D, aconseguir una representació contínua a partir d'aquest mostreig discret és una tasca complexa.

Els mètodes de reconstrucció de superfícies són els encarregats de tractar aquest problema per tal de recuperar una superfície continua que representi l'objecte en forma d'una malla de triangles, que facilitarà l'aplicació de tècniques de visualització i el processat posterior d'aquestes dades. Aquesta tesi proposa estratègies i solucions per tractar el problema de la reconstrucció de superfícies a partir d'un conjunt de punts.

La tesi comença per revisar l'estat de l'art dels mètodes de reconstrucció de superfícies. D'aquesta revisió, n'hem extret conclusions referents als defectes dels mètodes actuals, especialment quan aquests s'apliquen a núvols de punts creats a partir d'imatges subaquàtiques, ja que aquest tipus d'imatges presenten normalment soroll i outliers. Tenint en compte aquests problemes, en aquesta tesi proposem un seguit de mètodes que tenen en compte els errors comuns en aquests mostrejos, i promovem la creació de mètodes resistents tant a soroll com a outliers. Tanmateix, els nostres mètodes permeten recuperar superfícies amb llindars, fenòmen que apareix normalment quan s'explora un escenari submarí. A més, lluny de restringir la nostra àrea d'aplicació, assegurem que els algoritmes presentats no requereixen de cap altre tipus d'informació addicional, com ara les normals que necessiten molts dels mètodes de l'estat de l'art. Així, els mètodes proposats en aquesta tesi poden ser utilitzats amb qualsevol classe de dades basades en punts, sense tenir en compte la font d'aquestes.

Aquest treball contribueix a l'àrea de reconstrucció de superfícies amb quatre mètodes diferents. El primer es basa en modificar un mètode de mallatge existent. Aquest algoritme de mallatge de superfícies requereix una aproximació de la superfície que sigui capaç de proporcionar resposta a consultes d'intersecció entre un segment aleatori i la superfície

en sí. La nostra contribució consisteix en respondre aquest test d'intersecció sense tenir l'aproximació de l'objecte, és a dir, en respondre-la directament utilitzant el núvol de punts. Així doncs, donat el segment pel qual s'ha de trobar la intersecció amb l'objecte, es construeix una superfície local utilitzant els punts localitzats a una distància concreta d'aquest segment, de manera que es pugui utilitzar per trobar la intersecció entre aquesta superfície i el segment. A més, el mètode és capaç de construir aquestes superfícies tenint en compte els outliers i el soroll, d'escales variables, que poden estar presents en el conjunt de punts.

El segon mètode, que també treballa de forma local, construeix una representació intermitja de l'objecte anomenada representació de splats. Aquesta representació descriu l'objecte mitjançant superfícies locals d'extensió limitada generades a partir de cada punt. Tanmateix, aquestes superfícies locals es construeixen posant èmfasi en descartar outliers i atenuar el soroll. Aquesta aproximació inicial de la superfície és mallada en un segon pas, utilitzant una estratègia dissenyada específicament per aquesta nova abstracció. Això permet que la fase de mallatge, la qual defineix la qualitat de la malla de triangles final, s'executi amb diferents parametritzacions sobre la mateixa aproximació de l'objecte.

Els últims dos mètodes introdueixen una nova forma de fer, ja que canvien la visió local dels mètodes previs per una estratègia volumètrica global. Aquests dos mètodes es basen en construir una funció de distància sense signe, avaluada de forma discreta en una graella de tetraedres adaptativa, i definida a partir de la nostra representació de splats. Per tal d'extreure la superfície a partir d'un volum, l'espai s'ha de partir en dos, de manera que la superfície d'interès sigui la que separa aquests dos volums. Bàsicament, els dos mètodes proposats difereixen en com es realitza aquesta partició, tot i que ambdós es basen en una definició en forma de graf similar derivada de la graella de tetraedres prèviament mencionada. Per una banda, proposem un mètode que utilitza l'algorisme de talls S-T (S-T cuts) per separar la part interior de la exterior de l'objecte. Per altra banda, l'altre mètode utilitza l'algorisme de talls normalitzats (Normalized cuts) per partir el volum només utilitzant els valors de la funció de distància sense signe. Aquests dos mètodes aconsegueixen una millora en la resistència al soroll respecte els mètodes anteriors.

Els mètodes proposats són validats mitjançant la seva aplicació a conjunts de dades provinents de fonts varies i amb una àmplia diversitat en les condicions de mostreig. A més, els resultats obtinguts pels nostres algoritmes són avaluats i comparats tant qualitativament com quantitativament contra altres mètodes de l'estat de l'art.

Resumen

Los mapas del fondo marino son una fuente de información fundamental para la comunidad científica, dado que la cartografía del fondo marino es el punto de partida de la exploración de los océanos. Los avances en las metodologías de escaneado, utilizando técnicas tanto acústicas como ópticas, permiten que la construcción de mapas del fondo marino se realice a resoluciones cada vez más altas. Aun así, todas estas técnicas muestrean la superficie del área de interés obteniendo una nube de puntos. En el caso de áreas que presentan un gran relieve en 3D, conseguir una representación continua a partir de este muestreo discreto es una tarea compleja.

Los métodos de reconstrucción de superficies son los encargados de tratar este problema, recuperando una superficie continua que represente el objeto reconstruido en la forma de una malla de triángulos, que facilitará la aplicación de técnicas de visualización y el posterior procesado de estos datos. Esta tesis propone estrategias y soluciones para tratar el problema de la reconstrucción de superficies a partir de un conjunto de puntos.

La tesis empieza revisando el estado del arte de los métodos de reconstrucción de superficies. De esta revisión, hemos extraído conclusiones referentes a los defectos de los métodos actuales, especialmente cuando éstos son aplicados a nubes de puntos provenientes de imágenes subacuáticas, dado que éstas presentan normalmente ruido y outliers. Teniendo en cuenta estos problemas, en esta tesis presentamos un conjunto de métodos que tienen en cuenta los errores comunes en estos muestreos, y promovemos la creación de métodos resistentes tanto a ruido como a outliers. Así mismo, nuestros métodos permiten recuperar superficies con bordes, fenómeno que aparece normalmente cuando se explora un escenario submarino. Además, lejos de restringir nuestra área de aplicación, aseguramos que los algoritmos presentados no requieren de ningún otro tipo de información adicional, como las normales que necesitan muchos de los métodos del estado del arte. De esta forma, los métodos propuestos en esta tesis pueden ser utilizados con cualquier clase de datos basados en puntos, sin tener en cuenta la procedencia de los mismos.

Este trabajo contribuye al área de la reconstrucción de superficies con cuatro métodos diferentes. El primero se basa en modificar un método de mallado existente. Este algoritmo de mallado de superficies requiere una aproximación de la superficie que sea capaz

de responder a consultas de intersección entre un segmento aleatorio y la superficie en si. Nuestra contribución consiste en responder a este test de intersección sin tener una aproximación del objeto, es decir, en responderla directamente usando la nube de puntos. Así pues, dado el segmento para el cual se requiere la intersección con el objeto, construimos una superficie local utilizando los puntos que se hallan a una distancia concreta de este segmento, de manera que esta pueda ser utilizada para encontrar la intersección entre dicha superficie y el segmento. Además, el método es capaz de construir estas superficies teniendo en cuenta los outliers y el ruido, de escalas variables, que puedan estar presentes en el conjunto de puntos.

El segundo método, que trabaja también de forma local, construye una representación intermedia del objeto llamada representación de splats. Esta representación describe el objeto mediante superficies locales de extensión limitada generadas a partir de cada punto. Así mismo, estas superficies locales se construyen priorizando el descarte de outliers y la atenuación del ruido. Esta aproximación inicial de la superficie es mallada en un segundo paso, utilizando una estrategia diseñada específicamente para esta nueva abstracción. Esto permite que la fase de mallado, la cual define la calidad de la malla de triángulos final, sea ejecutada con diferentes parametrizaciones sobre la misma aproximación del objeto.

Los últimos dos métodos introducen una nueva metodología, ya que cambian la visión local de los métodos previos por una estrategia volumétrica global. Estos dos métodos se basan en construir una función de distancia sin signo, evaluada de forma discreta en una malla de tetraedros adaptativa, y definida en base a nuestra representación de splats. Para extraer la superficie a partir de un volumen, el espacio debe partirse en dos, de forma que la superficie de interés sea la que separe estos dos volúmenes. Básicamente, los dos métodos propuestos difieren en cómo se realiza esta partición, aunque los dos se basen en una definición en forma de grafo similar, derivada de la malla de tetraedros previamente mencionada. Por un lado, proponemos un método que utiliza el algoritmo de cortes S-T (S-T cuts) para separar la parte interior de la exterior del objeto. Por otro lado, el otro método utiliza el algoritmo de cortes normalizados (Normalized cuts) para partir el volumen utilizando solamente los valores de la función de distancia sin signo. Estos dos métodos consiguen una mayor resistencia al ruido que los métodos anteriores.

Los métodos propuestos son validados mediante su aplicación a conjuntos de datos provenientes de fuentes varias y con una amplia diversidad en las condiciones de muestreo. Además, los resultados obtenidos por nuestros algoritmos son evaluados y comparados, tanto cualitativamente como cuantitativamente, contra otros métodos del estado del arte.

Abstract

Underwater maps are an important source of information for the scientific community, since mapping the seafloor is the starting point for underwater exploration. The advance of range scanning methodologies, both using optical and acoustic techniques, enable the mapping of the seabed to attain increasingly larger resolutions. However, all these techniques sample the surface of an interest area in the form of a point cloud. For the case of areas containing non-trivial 3D relief, achieving a continuous representation from this discrete sampling is a complex task.

Surface reconstruction methods try to tackle this problem by recovering a continuous surface representing the object in the form of a mesh of triangles, easing visualization and further processing. This thesis proposes strategies and solutions to tackle the problem of surface reconstruction from point sets.

We start by reviewing the state of the art on surface reconstruction methods. From this survey, we extract some conclusions regarding the flaws in current methods, especially when applied to point clouds coming from underwater imagery which often suffer from noise and outliers. Taking these into account, in this thesis we devise a set of methods where common errors in the scanned measurements are taken into account by promoting methods resilient to both noise and outliers. Moreover, our methods allow the recovery of bounded surfaces, that usually appear when exploring an underwater scenario. Additionally, and instead of restricting ourselves to our application area, we ensure that the presented algorithms do not require any kind of additional information to work so they can be used with any kind of point-based data regardless of its source.

This thesis contributes to the area of surface reconstruction with four different methods. The first method is based on modifying an already existing surface meshing methodology. This surface meshing algorithm requires an already existing approximation of the surface able to provide segment-object intersection queries. Our contribution is to answer these intersection queries not by having the approximation of the object, but directly from the point set. Thus, given a required segment to be tested for intersection with the object, a local surface is built using the points falling at a given distance from this segment, so that it can be tested later for intersection with the segment. Furthermore, our method is able

to build these surfaces by taking into account the outliers and variable noise scale that may be contained in the point set.

The second method, also working in a local manner, constructs an intermediate representation of the object, called the splat representation. This representation describes the surface with small local patches of limited extent generated from each point. Again, these local surfaces are built focusing on disregarding outliers and attenuating the noise. This initial surface approximation is later meshed in a second step, using an approach tailored to this new abstraction. This allows the meshing phase, defining the quality of the final triangle mesh, to be executed with different parameterizations on a common approximation of the object.

The last two methods introduce novel thinking by changing the local view of the previous approaches to a global volumetric approach. The two methods are based on building an unsigned distance function, discretely evaluated on an adaptive tetrahedral grid, and defined from our splat representation. In order to extract a surface from a volume, the space should be partitioned into two, the surface of interest being at the interphase between these two volumes. Basically, these two methods differ in this partitioning procedure, even though both are based on a similar graph definition derived from the previously mentioned grid. On the one hand, we propose a method using S-T cuts to separate the inside and outside of the object. On the other hand, we use a Normalized cut approach to partition the volume using only the values of the unsigned distance function. The advantage of these two methods is that they achieve further noise resilience than the previous methods.

We validate the methods proposed through their application to datasets from various sources with a wide range of sampling conditions. Additionally, the results obtained by these algorithms are discussed and compared qualitatively and quantitatively with other state-of-the-art approaches.

Chapter 1

Introduction

1.1 Motivation

Despite the fact that almost every piece of land on earth has been explored, the conquest of the oceans still poses one of the greatest challenges for mankind. More than 70% of the surface of our planet is covered by water, and the hostile underwater environment prohibits humans from exploring at depths beyond the limits of scuba diving. This means that we know very little about a large part of our planet. Over the few last decades, advances in underwater robotics have opened the door to deep underwater exploration. Remotely Operated Vehicles (ROVs) as well as Autonomous Underwater Vehicles (AUVs), along with the information they gather, have benefited a large number of disciplines, biology, geology and archeology being some of the most relevant examples.

Shaping the otherwise unreachable underwater areas using the remote sensing provided by these platforms is of great interest to the scientific community. Underwater vehicles are equipped with a wide variety of sensors providing useful data for both navigation and mapping. The navigation and mapping problems are intrinsically related to one another: while navigation deals with the problem of knowing where the vehicle is at each step, the mapping process gathers all the local data collected by the sensors into a global representation that eases its interpretation. Mapping techniques are consequently divided according to the type of global representation they use, i.e., their mapping modality. These modalities can be broadly divided as follows:

- **2D maps**, which present the scene as being *flat*, useful in describing large areas compactly. From ancient times, cartography has used this representation in order to shape the world. Thus, having the collected data in a georeferenced 2D map allows correlating the retrieved data with many other maps made in the past. Of course, the main limitation of this kind of map is its inability to represent 3D relief.

- **2.5D maps**, also known as heightmaps/heightfields, represent the scene as a bivariate function so that, for each coordinate (x, y) in the plane, a given value of height z is provided, thus defining the injective function $f(x, y) = z$. When mapping the seabed, this representation is called a bathymetry. Of course, this kind of map can be represented as a 2D map just by giving a color to each of the z in $f(x, y)$. However, we refer to the 2.5D maps when height can be imaged in 3D using common visualization techniques. Despite being more informative than a pure 2D representation, this mapping modality is not able to represent arbitrary 3D geometry and, for instance, complex structures containing concavities cannot be represented.

- **3D maps**, the most versatile mapping representation, as it assumes no restriction on the shape of the object. In this case, the surface is normally modelled using a triangle mesh, allowing its visualization in modern graphics hardware, as well as easing its further manipulation using mesh processing techniques.

From the large sensor set the underwater vehicles are equipped with, optical cameras are one of the most versatile tools for mapping. It is widely known that the underwater environment poses some specific challenges to optical imaging. First, there is a rapid attenuation of the light in water, which causes the visibility to be restricted to a short range from the camera. Second, the reflection of light against suspended particles in the media may induce blurring and high frequency noise caused by the phenomenons of forward and backward scatter. Finally, the illumination of the images might be non-uniform because of the use of artificial lighting in deep water, or the sun flickering effect in shallow water. Some examples of the presented problems can be seen in Figure 1.1.

In spite of these challenges, the straightforward interpretation of the visual information collected using optical sensors posed them as a relevant tool in underwater surveying. In order to alleviate some of these challenges, the data collection must be carried out at a very short range to the surveyed scene, which translates in having a very local portion of the scene mapped in a single image. In order to get a comprehensive view of the entire scene, these local contributions have to be merged in a global representation, i.e, the map. Noting again the correlation between navigation and mapping, it is possible to build a global map as long as some knowledge of the global or relative camera positioning/trajectory exist. The motion followed by the camera from frame to frame may be computed using computer vision techniques, which may additionally be helped by other sources of navigation information provided by further sensors. Finally, the process of transforming the set of images into a common representation is called the reconstruction process. Most of the reconstruction techniques are feature-based, meaning that they use in their computations only one relevant subset of points in each image to compute both the motion and the

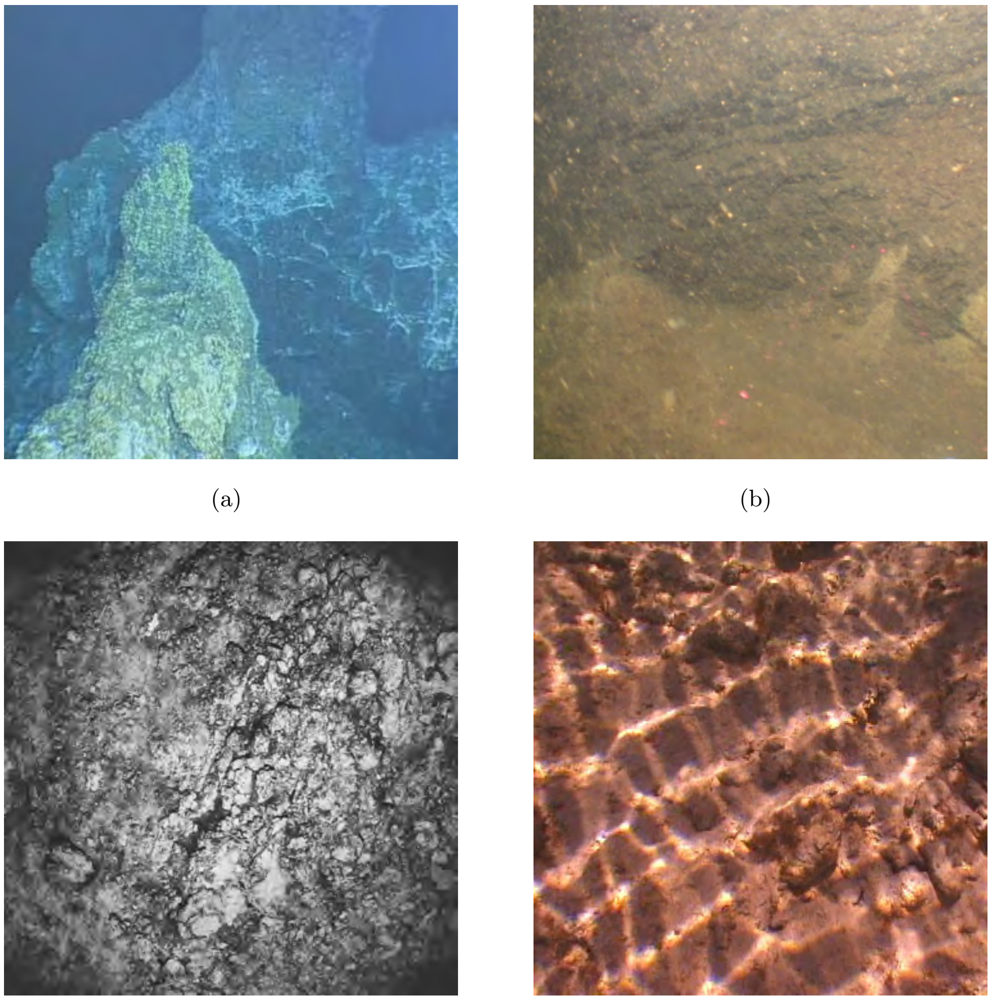


Figure 1.1: The challenges of underwater imaging: forward scattering on (a) causes the image to be blurred, while backward scattering on (b) causes the image to be noisy, presenting these *white dots* corresponding to suspended particles in the water. Additionally, the uneven illumination caused by using an artificial light source in deep water surveys is shown in (c), while in (d) one can see the illumination pattern caused by sun flickering in shallow water. Note that (a) also presents the effect of light attenuation with distance, as objects closer to the viewer preserve their colors better than those at the back.

desired structure of the scene. These feature points are basically distinctive 2D points in the image, according to their surrounding texture. These feature points are also matched across images, so that knowing the one-to-one relationships between the points in the image pairs in the sequence allows the computation of the camera's motion.

For the 2D case, also known as photomosaicking, the estimated camera trajectory is used to map the original images onto a common plane (the final map). The planarity assumption of the scene allows the transformations between frames to be represented by homographies, and pose the problem with fewer unknowns [78]. In the 2.5D/3D case, the motion model has to be more general, and the complete motion (rotation and translation) is computed for each frame in the sequence of images. It is worth noticing that photomosaics are widely used in underwater applications for monitoring areas ranging from small (Gracias et al. [93]) to large (Barreyre et al. [19]) scale.

In the 2D case, there is a base plane onto which the images can be projected, i.e., once the motion is extracted, thus building the map mainly consists of warping and deforming the images onto this common plane. However, in the 3D case, the problem becomes more complex, as this base representation is missing. Once the trajectory of the camera has been recovered, the 3D positions of the points can be extracted by triangulating in 3D the line-of-sight rays emerging from each camera-feature pair. This process results in a point set representation of the object. For the 2.5D, the problem of finding a base representation is solved by assuming that our scene can be modelled using a heightmap, that is, the mapping is simplified at the expense of loosing the ability to represent complex details. The common technique is to find a common plane, as in the 2D case, and project all the 3D points onto it. Then, these points in 2D are triangulated using some technique and the texture can be projected into these triangles by using some heuristics. This triangulation is then *lifted* back to 3D by adding the points' z coordinate, resulting in a pseudo-3D representation.

As previously stated, the method above cannot capture fine details in complex 3D structures. For this purpose, a more detailed representation of the surface of the surveyed object/scene is needed. The problem is then to find a piecewise linear approximation of the shape of the object given the sample points. As for the 2.5D case, the most used representation is a mesh of triangles. This problem is known as the surface reconstruction problem from an unorganized set of points (in the following referred to as the surface reconstruction problem), since this point set does not follow any regular lattice, and the result is a triangle mesh following the surface defined by the points. Note however, that the surface reconstruction problem is ill-posed, as illustrated in Figure 1.2: for a given set of points, one can define several surfaces that might have generated the sampling. Moreover, the problem can be solved in two ways basically. One may adopt an interpolation strategy,

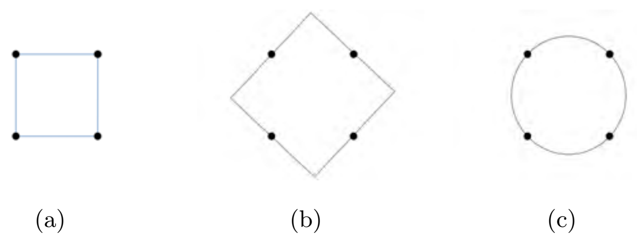


Figure 1.2: Illustrative example of the surface reconstruction problem being ill-posed. For easiness of depiction, the example is a curve reconstruction, the equivalent problem in 2D. Note how, given the set of points shown in black, one may define several curves passing on or near the points that might have generated this sampling.

which can be seen as a connect-the-dots game. That is, the neighborhood relationships in 3D have to be found in order to build triplets of points forming the triangles of the surface. This approach works well when the data is not corrupted by noise, that is, the points are considered to be measures taken *exactly* on the surface of the object. In the case of noisy input, the points might be seen as a *notion* of where the surface should be, but knowing that the points may not be exactly on the surface. The problem is then to get a triangle mesh that approximates the shape of the object from the points, without these points forming part of the final set of vertices on the mesh.

As a matter of fact, most present-day scanning devices (being optical or not) provide the scanned object in the form of a point cloud. These scanning technologies may include, among others, range scans, widely used in land robotics, or (multibeam) sonar systems, used underwater. However, the point set representation has two main drawbacks. On the one hand, it is difficult to interpret and visualize. When these points are shown on screen, it is almost impossible to tell which points are on the front, and which are at the back (see Figure 1.3), in other words, since the points are infinitesimally small, from any given viewpoint we cannot tell which part of the object should be visible and which should be occluded. On the other hand, working with points alone makes further computations complex to apply to the point set directly. Many processing techniques rely on knowing the relationships between those points. Simply stated, we are not able to tell the area of the surveyed object from the points alone. Even if some approximations may be used to work directly on point set data, the computations are far simpler when the continuous surface of the object is known.

Thus, given its wide application, and the lack of solutions in the literature regarding robust methods handling this issue, in this thesis we focus on proposing methods to solve the surface reconstruction problem.

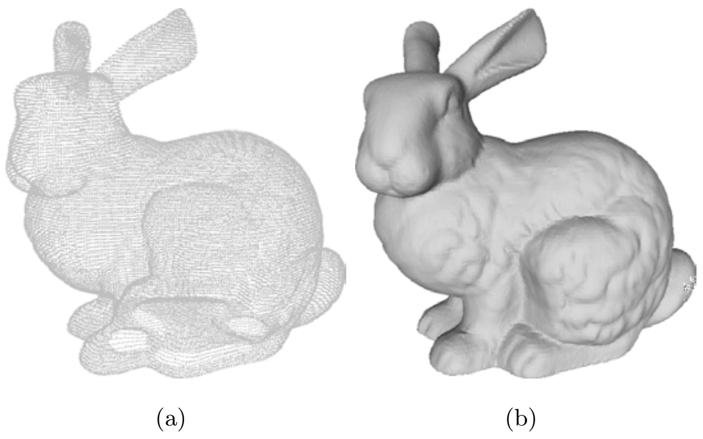


Figure 1.3: The visibility problem on point sets. Given a set of 3D points projected on a screen (a), one is not able to distinguish at a first sight, which of the points are on the front part and which at the back, precluding like this the interpretation of the 3D shape of the object. On the other hand, if we have a representation of the surface as we do in (b), the shape of the object is clearly distinguishable.

1.2 Objectives

The goal of this thesis is to develop methods to solve the **surface reconstruction problem** to help in filling the existing gap in present-day 3D optical mapping techniques. More specifically, given a possibly **noisy set of points** sampled at the surface of an object, we aim at recovering its shape as a surface mesh of triangles. This representation of the object allows the development of virtual environments useful for applying remote processing and calculations of the scene after surveying.

Despite the main focus of this thesis being the exploration of the oceans, the presented methods are generic enough to **work on raw point sets**, thus being applicable to a wide variety of input data. The only requirement is knowing the surface through a set of points sampled on (or near) the surface of the surveyed object, without any additional information assumed. This loose requirement allows using data coming from sources other than optical. We have successfully proven this versatility by applying the methods in datasets ranging from in-lab object reconstruction to out-door natural and urban scenes.

Furthermore, we focus on developing robust methods that **allow the input data to be corrupted with both noise and outliers**. In our precise scenario, noise refers to the repeatability of the sensor (i.e., how accurate the measures are), while outliers are erroneous measurements that should be ignored when building the surface. These phenomenons appear with more or less importance in all real-world datasets, but very few methods in the state of the art try to deal with both problems simultaneously.

Additionally, the mapping of the seabed gives rise to open surfaces. State-of-the-art approaches have mainly focused on recovering closed surfaces, as this imposes some restrictions that eases the reconstruction problem when cast to a volumetric view. In our case, we **recover bounded surfaces** by not reconstructing parts of the surface away from the samples.

1.3 Contributions

The main contributions of the present thesis can be summarized as follows:

- A novel method which meshes the surface by locally building small surfaces to answer the queries required by a Delaunay refinement-based surface meshing algorithm.
- A two-step method where an intermediate splat representation is created as an approximation of the shape, which is then meshed to obtain the final surface.
- Two volumetric methods extracting the surface at the interphase of two volumes, where these volumes are the result of partitioning an unsigned distance function using graph-based techniques.

When compared to the state of the art, our methods achieve the following improvements:

- Raw input, i.e., just 3D points are required in order to extract the surface, without any additional information.
- Work on corrupted datasets, dealing with both noise and outliers in the data, by means of using techniques from the robust statistics literature.
- Recovery of bounded surfaces.
- Coarse-to-fine approach, allowing the user to define the stopping criterion based on the quality assessment of the final surface in terms of approximation error, size and shape of its triangles.

We thoroughly develop the contributions of each of the proposed methods in their corresponding chapters, and summarize them again in Chapter 9.

1.4 Organization of the Document

This thesis is organized as follows:

- **Chapter 2:** Covers the optical 3D reconstruction pipeline, explaining how the datasets used in subsequent sections are obtained.

- **Chapter 3:** Describes some geometrical concepts and data structures that help in understanding the foundations of the methods presented in Chapter 4.
- **Chapter 4:** Contains a thorough review of the state of the art in surface reconstruction, allowing the detection of flaws and strengths in present-day approaches which motivate the creation of the methods presented in Chapters 5, 6 and 7.
- **Chapter 5:** Presents our first proposal for surface reconstruction working on raw point sets. Local surfaces are built on demand in order to answer the intersection query required by a Delaunay refinement surface mesher algorithm. We include a description of the method and validate its application to different kinds of data.
- **Chapter 6:** We extend the previous proposal to add an intermediate step consisting of a splat-based representation of the object. At a second step, this splat representation is meshed by adapting a Delaunay refinement step. As for the previous approach, this section contains a description of the method, its insights, and its evaluation with various datasets.
- **Chapter 7:** The local contributions provided by the splat representation are merged into a global unsigned distance function. In order to extract the surface from the volumetric representation, the distance function is signed following two proposals using minimum cuts in graphs induced from the space partition defining the distance function. Again, we test the methods on a wide variety of input point sets.
- **Chapter 8:** We provide a benchmark evaluation that quantitatively puts all the algorithms presented in the thesis in context with the state of the art. Furthermore, the comparison in the original article is extended with even more methods.
- **Chapter 9:** Finally, this document ends with a dissertation on the contributions and further works that can derive from the present thesis.

Chapter 2

3D Optical Modelling

2.1 Introduction

Recovering the shape of an object from a set of images has been one of the earliest and fundamental problems in computer vision. This chapter presents an overview of the 3D optical reconstruction process, with special emphasis on feature-based approaches. The results obtained by these methods normally present the reconstructed model as an unstructured point set, where each point is a discrete sample of the surface of the object of interest. This information is used in further steps of the present thesis to recover this surface. This chapter is thus intended to give an overview of the creation of 3D models from images as a whole, clearly spotting our work inside the processing pipeline. Additionally, at the end of this chapter, we also present how 3D modelling has been used in underwater applications in recent years, and how our contribution can help in developing new ways for highly detailed seabed exploration.

2.2 Overview

There have been many variate approaches by the computer vision community dealing with the automatic reconstruction of an object or a scene in 3 dimensions. Nevertheless, we do not aim to present a review of the state of the art in 3D modelling. On the contrary, this chapter is tailored to understand the basis of the approach applied to obtain the datasets used in our experiments. Consequently, we omit many other existing approaches that are equally valid, but not used on our work. For example, volumetric reconstruction approaches (voxel coloring [187] and similar approaches [130, 179, 24, 195, 213]) have also shown to perform well in controlled environments, but they will not be covered here as they go beyond the scope of this thesis.

Looking at the pipeline as a whole, the reconstruction process can be divided into sequential stages, where the output information of one is used as input for the following. Even though several variants have been proposed in the literature, we pose the modelling pipeline as follows:

1. **Camera calibration:** Consists of obtaining the intrinsic parameters ruling the internal geometry of the camera, i.e., how a point from the world is projected onto the image plane.

2. **Correspondence search:** Extracts a subset of points from the images and finds the correspondences between them over different views. The correspondence between points is obtained by the detection of features, which are those points having a relevant variation in their surrounding texture. The description of this texture, is then used to compare the similarity between features.

3. **Structure from motion:** In order to recover a 3D object, it is important to recover the position from which each image was taken. The motion of the camera and the shape of the object are intrinsically related. On monocular camera systems, the algorithms responsible for getting this kind of information are referred to as structure-from-motion methods. In addition to the camera's trajectory, these methods recover a sparse point set representing the object. These points are relevant image features which are then matched and tracked through the process.

4. **Dense reconstruction:** Despite the fact that methods from the previous step already provide reconstruction of the feature points, this representation does not sample the object densely enough. For this reason, the dense reconstruction step tries to increase the number of points describing the object by using a further correspondence search.

5. **Surface reconstruction:** The points found in the previous step represent samples taken from the surface of the object to be reconstructed. Despite not being the only available surface representation, we aim at obtaining a triangle mesh. The advantages of this representation compared with others will be discussed in Chapter 4.

6. **Post-processing:** Given the reconstructed surface mesh, this structure can be manipulated using any of the mesh processing techniques in the literature. Furthermore, more related to the optical origin of the source data, texture mapping processes are desirable in order to provide realistic models.

In the following we develop each of the steps in more detail, with the exception of the

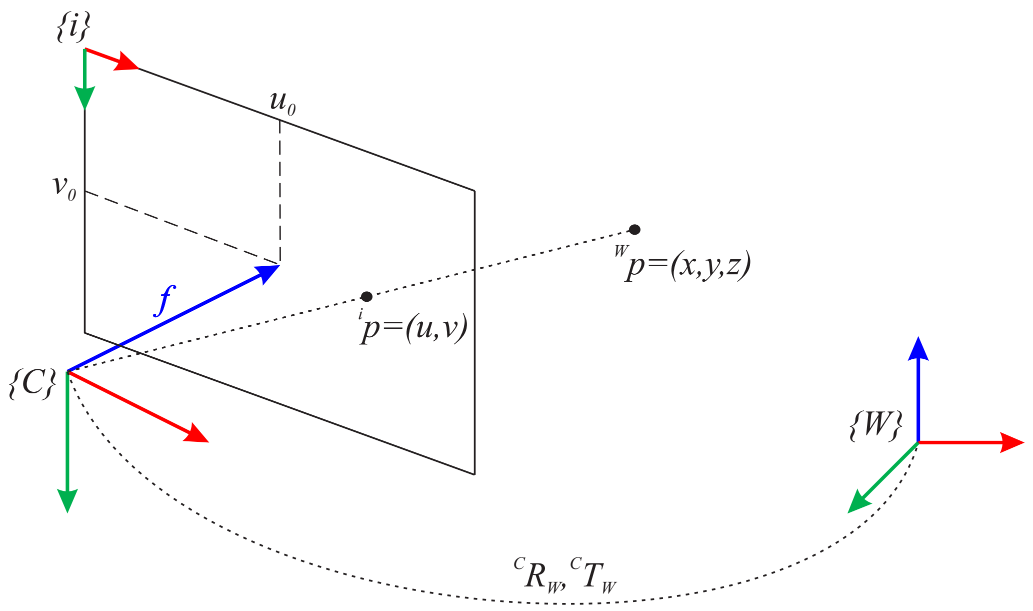


Figure 2.1: Depiction of the projection of a 3D point onto a 2D image plane. Regarding the axis colors: Red/Green/Blue denote the X/Y/Z axis respectively.

surface reconstruction issue, which will be thoroughly detailed in further chapters of this thesis.

2.3 Camera Calibration

In order to recover the original 3D position of a point, the first requirement is to know how the projection onto the 2D plane takes place. For the sake of dealing with simple transformations, the pinhole model is assumed, where the projection onto the image plane is greatly simplified by assuming that all the light rays pass through the center of the camera and are projected onto the image plane at a distance restricted by the focal length f. Despite the simplicity of this model, it is the most widely used nowadays [103]. Figure 2.1 shows the projection geometry: given a point ${}^{W}p = (x, y, z)$ in world coordinates, its 2D projection onto the image plane ${}^{i}p = (u, v)$ of the camera located at C is given by the following expression:

$$\delta \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = {}^{i}\Pi_W \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \tag{2.1}$$

where ${}^{i}\Pi_W$ is the 3×4 projection matrix, passing from the $\{W\}$ coordinate frame to the image one $\{i\}$, and δ is a scaling factor. The projection matrix contains both the extrinsic and intrinsic parameters:

$$^{i}\Pi_{W} = K\left[\ {}^{C}R_{W}\ \middle|\ {}^{C}T_{W}\ \right]. \tag{2.2}$$

On the one hand, the extrinsic parameters refer to the pose of the camera, i.e. rotation ${}^{C}R_{W}$ and translation ${}^{C}T_{W}$ with respect to the world reference frame $\{W\}$. On the other hand, the intrinsic parameters model the transformation K between the camera reference frame $\{C\}$ and the projection into the image frame $\{i\}$:

$$K = \begin{pmatrix} f \cdot s_u & f \cdot s_s & u_o \\ 0 & f \cdot s_v & v_o \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.3}$$

The K matrix is the *calibration matrix*, where f represents the focal length, which is the orthogonal distance from the center of the camera to the image plane, and (u_0, v_0) represent the principal point, which is the point located at the intersection of the focal axis and the image plane. Additionally, the parameters s_u and s_v represent the relationships between world units and pixels (pix/mm) along the x and y axis of the image respectively. Finally, s_s describes a skew parameter, only used in cases where the axes of the images are not fully orthogonal.

Note that the pinhole model is not able to represent fully the distortions induced by optical systems in real cameras. Figure 2.2 presents the two common distortions that are present on real optical systems. Obviously, this distortion is non-linear, and has to be corrected in order to apply the linear geometry posed by the pin-hole model. Given a 3D point ${}^{C}p$ in the camera reference frame, the normalized projection is as follows:

$$p_d = \begin{pmatrix} \frac{{}^{C}x}{{}^{C}z} \\ \frac{{}^{C}y}{{}^{C}z} \end{pmatrix} = \begin{pmatrix} x_d \\ y_d \end{pmatrix}. \tag{2.4}$$

Using the distortion model of Brown [38], this distorted position of the point in the real image is related with its undistorted version through radial and tangential terms. In fact, each of these distortion types theoretically requires an infinite series of correction terms. On the practical side, most applications work with just the first two terms of the radial correction, omitting the tangential one. A common model adopted by many authors is to correct up to the third term for radial distortion and up to the second term for tangential distortion:

$$p_u = \begin{pmatrix} x_u \\ y_u \end{pmatrix} = \left(1 + k_{r1} \cdot r^2 + k_{r2} \cdot r^4 + k_{r3} \cdot r^6\right) \cdot p_d + d_t, \tag{2.5}$$

where $r^2 = x_d^2 + y_d^2$ and d_t mark the tangential distortion vector defined as:

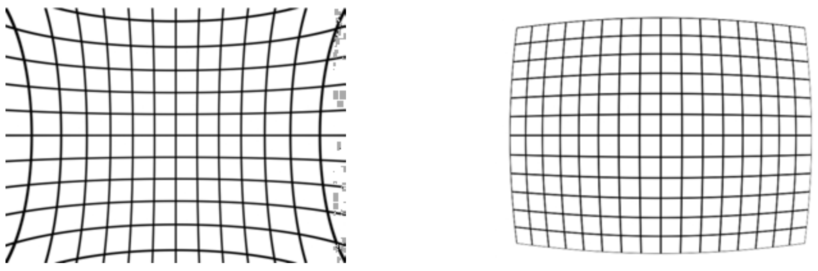


Figure 2.2: Common image distortion types introduced by optics on real-world cameras. On the left, pincushion distortion, on the right, barrel distortion. Note how the straight edges of a regular grid in the real world are imaged under both types of distortion.

$$d_t = \begin{pmatrix} 2 \cdot k_{t1} \cdot x_d \cdot y_d + k_{t2} \cdot (r^2 + 2 \cdot x_d^2) \\ k_{t1}(r^2 + 2 \cdot y_d^2) + 2 \cdot k_{t2} \cdot x_d \cdot y_d \end{pmatrix}. \tag{2.6}$$

Once the radial distortion coefficients k_{r1}, k_{r2}, k_{r3} and the tangential ones k_{t1}, k_{t2} are known, the coordinates of the undistorted point p_u finally allow the application of the linear relations induced by the camera matrix K, so that the image pixel coordinates of the point can be computed:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K \cdot \begin{pmatrix} x_u \\ y_u \\ 1 \end{pmatrix}. \tag{2.7}$$

The procedure to obtain the internal parameters is called the calibration of the camera. This calibration is usually performed using a calibration pattern, from which some points belonging to a common known plane can be extracted. There are many software tools available for this task, but the most commonly used is the one presented by Bouguet [35]. By means of a known planar pattern, and some user intervention, the 3D-2D relations can be established. Then, by means of the Direct Linear Transformation (DLT) [103] algorithm and a non-linear optimization procedure, the parameters of the camera are recovered. Given the tedious process of finding the 2D-3D correspondences, in the last few years there has been an increasing interest towards automatizing the calibration process as much as possible. These approaches are called self-calibration methods, and some examples can be found in Zhang [214] and Svoboda et al. [202].

Note that the camera calibration can be included inside the camera trajectory estimation (i.e., the calculations of the extrinsic parameters). The intrinsic parameters are then computed inside the pose estimation method, by making some assumptions about them. For example, in Snavely et al. [197] they assume the principal point as being the exact center of the image, $k_u = k_v = 1$ and $k_s = 0$, so that the problem is reduced to finding the focal length and the distortion parameters (which they compute to the second coefficient).

However, in order to alleviate the complexity of the problem, it is strongly advised to compute the intrinsic parameters using a calibration procedure before advancing to further steps in the reconstruction pipeline.

2.4 Correspondence Search

In the previous section, we have seen how a point can be projected into the image frame. However, our problem is posed in the reverse configuration: we have a 2D point in the image and we want to recover the 3D point that generated it. By taking a further look at Figure 2.1, it is obvious that the problem is not solvable using a single camera, since any point on the ray joining the center of the camera and the 3D point would project to the same position in the image. It is thus required to have two or more views of the same point in order to disambiguate the problem, as presented in Figure 2.3.

Basically, when the positions of the cameras are known, the 3D position of the points can also be recovered. Otherwise, when no camera pose information is available, the difference in location of one point when seen from distinct views gives us information about the motion the cameras have followed. Consequently, it is of great importance to be able to match the same point reliably over different images. This problem is called the correspondence search (or matching) problem, and is the base tool to recover both the trajectory of the camera and the shape of the object.

In this section, we overview the matching process, which can be divided into 3 main steps: detection, description and matching.

2.4.1 Detection

Trying to find a correspondence for each pixel in the image is, on the one hand, computationally expensive, and on the other, a very ambiguous problem, since depending on the characteristics of the texture in the images, some points may look very similar without being the same. Thus, the detection process aims at getting a subset of salient points in the image so that they are relevant according to their surrounding texture. In order to be able to match them correctly across images, they need to be repeatable, so they can be found again, and discriminative, so they can be correctly identified. From the huge amount of proposals available, we here give a brief overview of some of the most relevant feature detector methods, exemplifying their behaviour in Figure 2.4:

- **Harris corner detector:** Presented by Harris and Stephens [102], this is one of the most used feature detection methods. It detects corner points with a high gradient on both X and Y directions through an eigen analysis of the so-called autocorrelation

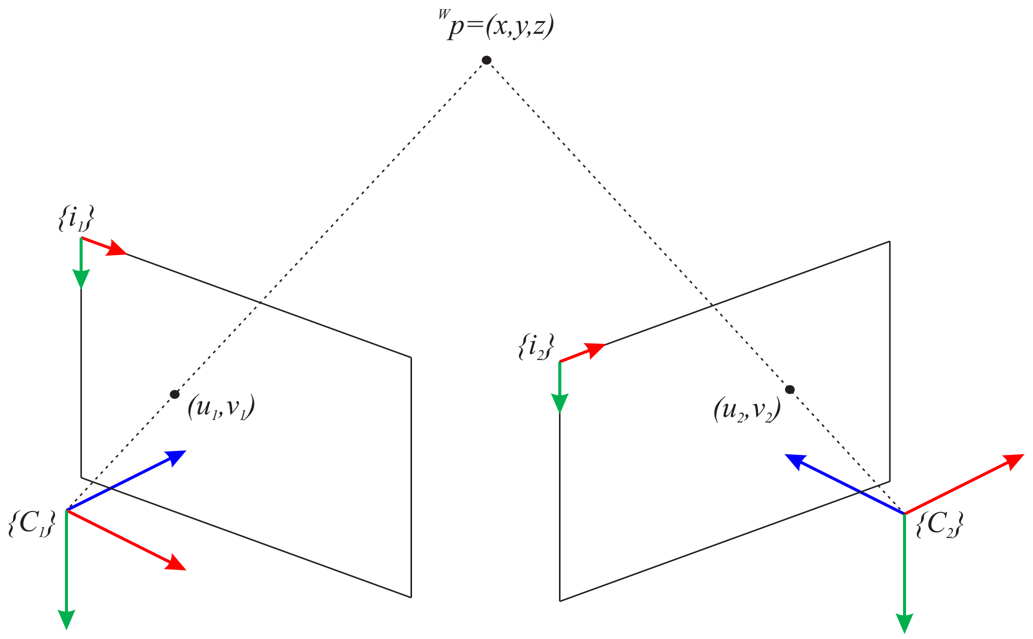
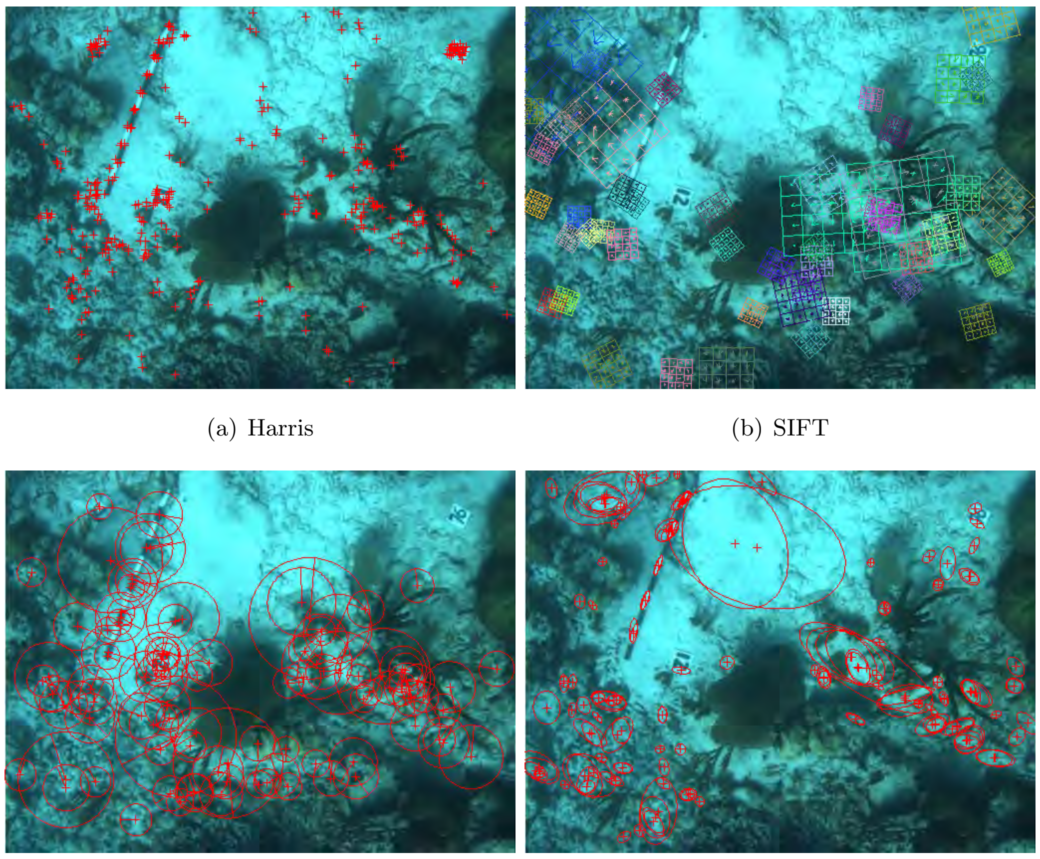


Figure 2.3: Stereo setup: two (or more) views disambiguate the 3D position of the point along the line-of-sight emanating from the camera's optical center and passing through the 2D point projection.

matrix for every pixel in the image.

- **SIFT:** The Scale-Invariant Feature Transform (SIFT) method presented by Lowe [147] extracts distinctive regions using an approximation of the Laplacian of Gaussian (LoG) by using the Difference of Gaussians (DoG). The image is convolved with different Gaussian kernels at increasing scales (the standard deviation of the Gaussian), and then the convolved images are grouped in octaves. These octaves correspond to doubling the value of the scale, and finally the DoG is obtained by subtracting two adjacent convolved images. The whole stack of DoG images form a scale space, from which keypoints are extracted as local extrema.

- **SURF:** Stands for Speeded Up Robust Features (SURF), presented by Bay et al. [20]. This method presents a pipeline similar to the one in SIFT, but decreasing its computational effort. In this case, the determinant of the Hessian matrix, applied in a scale space, is used to detect the features. Thus, the idea of a scale space is also applied, but modified to optimize the computational cost. This optimization starts by using box filters as an approximation of the second order derivatives of the Gaussian filter. These box filters, in conjunction with the use of integral images, provide fast evaluations. On the other hand, the scale space is created by modifying the size of the filter, thus avoiding the computational burden of scaling the size of



(a) Harris (b) SIFT

(c) SURF (d) MSER

Figure 2.4: Examples of feature detection algorithms applied to an underwater image. The methods tested are those commented on the text: (a) Harris, (b) SIFT, (c) SURF, and (d) MSER. The number of features has been deliberately reduced to promote illustration clarity.

the images, as required by SIFT.

- **MSER:** The Maximally Stable Extremal Regions (MSER) is a blob detection algorithm presented by Matas et al. [155]. It extracts regions having higher or lower intensity values than all their surrounding pixels. This is done using a series of binarizations, where the threshold used is gradually increased. The regions of pixels that have a low variance over a large range of these thresholds are the ones selected.

2.4.2 Description and Matching

Given two images, and using the methods presented above, we have two sets of relevant feature points. The matching phase consists of finding correspondences between points in these two sets. For this purpose, a description for each feature has to be provided. Thus, feature points need to be characterized, based on their surrounding texture, in a way that allows easy recognition in other images.

Again, there are many variants for this task, but the most commonly used work with template matching techniques or characterize texture using the surrounding gradients. Template matching consists of using the texture around a point as the information to describe it. Thus, some statistics like the Sum of Squared Differences (SSD) or the Normalized Cross-Correlation (NCC) are used on windows of fixed size around the points. The similarity values of these statistical measures are used in order to rank the goodness of a given match. These approaches tend to offer only invariance to translation and slight tolerance to a small rotation (< 5-$10°$) if circular patches are considered.

On the other hand, instead of computing the similarity at the intensity level directly, the gradients of this texture can be used to describe a feature. SIFT and SURF algorithms use this methodology in similar ways. We now summarize the histograms of gradients method used by SIFT. Inside its scale space, the descriptor is constructed at the scale where the feature was detected. The texture surrounding the point is divided into sub-windows of $n \times n$ pixels. In each of these sub-windows, the gradients are computed and a histogram of magnitudes is constructed by binning the gradients into 8 buckets covering the 360 degrees, and a dominant orientation is computed from it. The final description vector is a stack of 4×4 histograms of 8 bins each, composing a total of 128 elements. The procedure used in SURF uses a similar method, but the gradients are approximated using responses to Haar wavelets, and the final vector contains 64 elements. This algorithm, combined with the use of a scale space in SIFT/SURF detection, provides translation, rotation and scale invariance to the features.

Once a description vector is obtained for each point, their similarity can be computed as the Euclidean distance between them. However, in order to improve this simple check, the similarity with the second nearest neighbour can be also taken into account. In this case, a match is considered as *good* if the ratio between the distances to the first and second nearest neighbour is above some threshold, i.e., the first neighbour can be considered distinctive enough from the second.

Despite the efforts of the methods presented so far, the one-to-one matching process is still error prone, and many *outlier* matches (i.e., wrong matches) might be found by relying only on similarity measures. In order to improve the results, we have two options depending on the available data. On the one hand, when known extrinsic parameters are available (e.g., when using a stereo rig, or additional motion sensors in monocular sequences), they can be used to further restrict the search for correspondences. On the other hand, when no information about the motion between images is available, the a-priori assumption of the scene being rigid can be exploited to alleviate the number of false matchings.

2.4.2.1 Epipolar Constraints

When the extrinsic parameters of the camera are known, the geometry between cameras directly poses a set of restrictions that might be used to consequently discard some of the point associations found in the matching step. These restrictions are driven by epipolar geometry.

When a scene is seen from two different viewpoints, there are a series of relationships and restrictions between the 3D points and their 2D projections. These restrictions in the stereo case are called epipolar constraints. By using them, the search space at the time of matching is greatly reduced. These relations are easily interpretable by looking at Figure 2.5. The basis of epipolar geometry is the observation that any 3D point ${}^{W}p$ and its 2 projections on 2 images p_1 and p_2 are coplanar. Furthermore, this plane (named Δ in Figure 2.5) also contains the optical centers of the cameras c_1 and c_2. Suppose we have detected the point p_1 in the first image and we want to obtain its correspondence in the second one. Given the known extrinsic parameters, we know c_1 and c_2, which along with p_1 generate the plane Δ. Thus, it is obvious that the correspondence for p_1 in image 2 can only be found along the line generated by the intersection between Δ and the image plane. It is worth noticing that points e_1 and e_2, which are the so called epipoles, also represent the projection of the center of an image into the other (e_1 is the projection of c_2 into the first image and viceversa). Restricting the correspondence search along the epipolar line reduces the search space from bidimensional (the match could be anywhere on the image) to one dimension (the match is along the line). Of course, given the uncertainty of the calibrated extrinsic parameters, this search space might have to be constrained to a few pixels above or below the line.

In fact, all these relations can be encapsulated in a 3×3 matrix, called the Essential matrix in case of known intrinsics, or the Fundamental matrix otherwise. In the calibrated case, i.e., when the coordinates of p_1 and p_2 are described with respect to the camera frames, one can represent the coplanarity relationship between the camera centers and the matches p_1 and p_2 in this matrix form:

$$p_1(t \times R \cdot p_2) = 0. \tag{2.8}$$

The Essential matrix E then contains the cross product between rotation and translation in its matrix representation, using a skew symmetric matrix for the translation

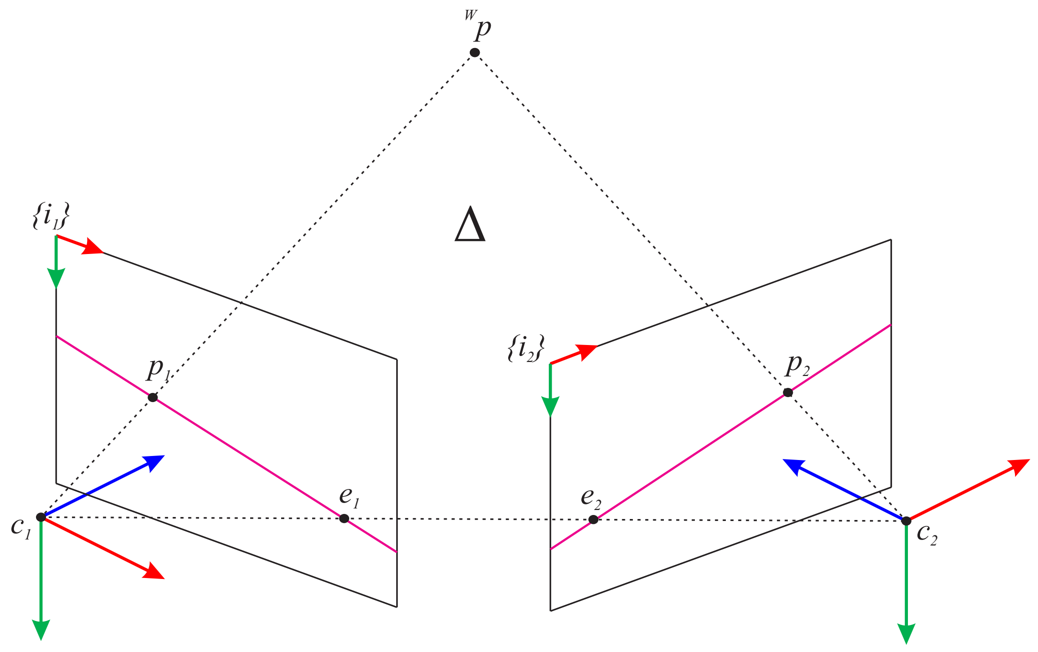


Figure 2.5: Epipolar geometry. The plane joining the 3D point ^{W}p with c_1 (resp. p_1) and c_2 (resp. p_2) defines the epipolar constraints. Basically, the match for p_1 can only be found along the epipolar line (pink segment in the image) in image 2 and viceversa.

part:

$$E = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \cdot R. \tag{2.9}$$

In the case of unknown intrinsics, the Essential matrix has to contain also the camera parameters, which gives rise to the Fundamental matrix:

$$F = K_2^{-T} \cdot E \cdot K_1^{-1}. \tag{2.10}$$

Obviously, if the geometry is known, the E/F matrixes help in restricting the search space. However, in the case of unknown extrinsics, the E/F matrices can be used as the unknowns of our problem, as the 9 parameters forming them (8 when forcing $F/E(3,3) = 1$) encapsulate all the geometry between frames.

2.4.2.2 Robust Matching

When no information other than the images is given, the knowledge of the observed scene being static is the only cue to exploit in order to alleviate false matchings (see Figure 2.6). Since the scene does not move, all the matches are restricted to following the same motion.

In order to force this restriction, a motion model is assumed, and robust statistical methods such as Least Median of Squares (LMedS) [177] or RANdom SAmple Consensus (RANSAC) [79] are used in order to ensure that all the selected matches follow the same motion. In fact, LMedS and RANSAC aim at obtaining a set of correspondences which is free of outliers according to the selected model. This model encodes the motion information. In this way, the matching and motion estimation processes are linked.

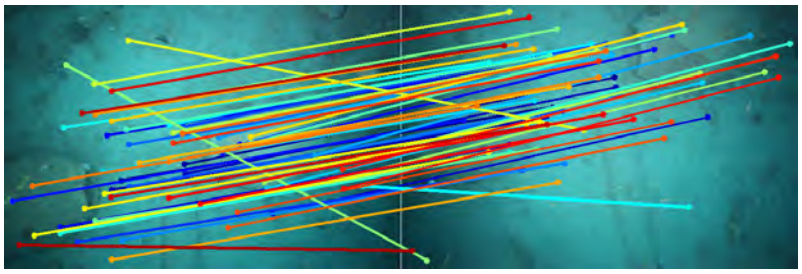
Both LMedS and RANSAC methods rely on stochastic sampling. At each iteration, a model is constructed using the minimum number of points possible. This model can be, for example, a homography or an essential/fundamental matrix. Then, from this candidate model, a measure of *goodness* is extracted. In the case of LMedS, the validity of the model is computed as the median of the residuals from all the matchings to this model. However, in the case of RANSAC, a number of votes is given to the points falling closer than a given distance from the model, in a way that models close to the ideal obtain more votes. Many candidates are selected iteratively, and after several steps the one obtaining the best consensus is retained as the final model.

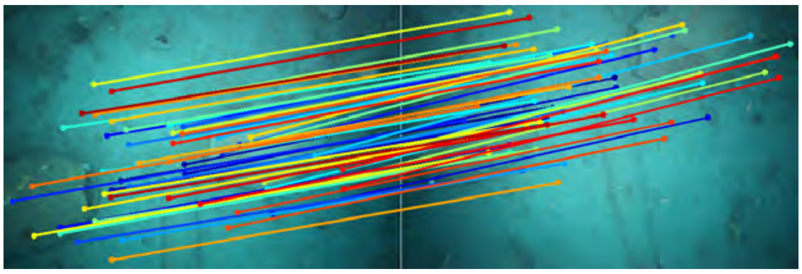
In the end, both epipolar geometry and robust matching aim at obtaining matches between image pairs. In the next section, we overview how to gather the pairwise information together in order to obtain the final reconstruction of the shape of the object as a point cloud.

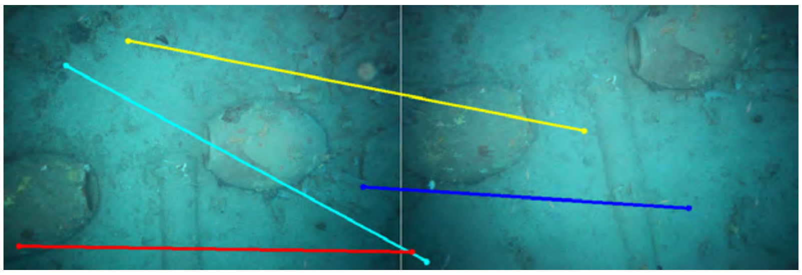
2.5 Structure from Motion

In calibrated environments, where both intrinsic and extrinsic parameters of the cameras used to capture the scene are known, the problem of computing the structure of the object is reduced to using lines-of-sight. Lines-of-sight are the rays that emanate from each camera center and pass through the image points. The 3D position of the point can be obtained by the process of triangulation, consisting of finding the intersection between the lines-of-sight generated from the projection of the same point in different views. As shown in Figure 2.7, due to errors in the camera's positioning, the lines-of-sight are not likely to intersect exactly. For this reason, the final intersection point is evaluated as the solution to the linear system of equations defined by the following restrictions:

$$\begin{array}{rcl} p_1 \times \Pi_1 \cdot {}^{W}p & = & 0 \\ p_2 \times \Pi_2 \cdot {}^{W}p & = & 0 \\ \cdots & & \\ p_n \times \Pi_n \cdot {}^{W}p & = & 0, \end{array} \qquad (2.11)$$


(a) Initial Matches


(b) Accepted Matches


(c) Rejected Matches

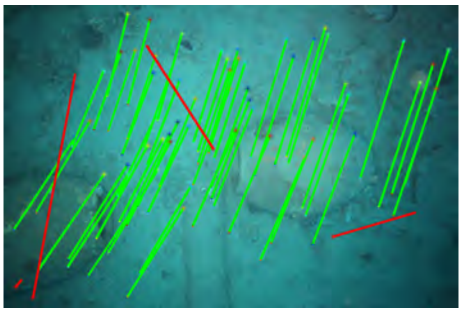

(d) Motion

Figure 2.6: Matching steps: (a) shows the result after the nearest neighbor search. Using RANSAC, and the fundamental matrix as the motion model, the accepted matches following the same motion are shown in (b), while the rejected ones are presented in (c). Finally, (d) shows the disparity of these features, i.e., the difference in position from one image to the other (green=accepted/red=rejected).

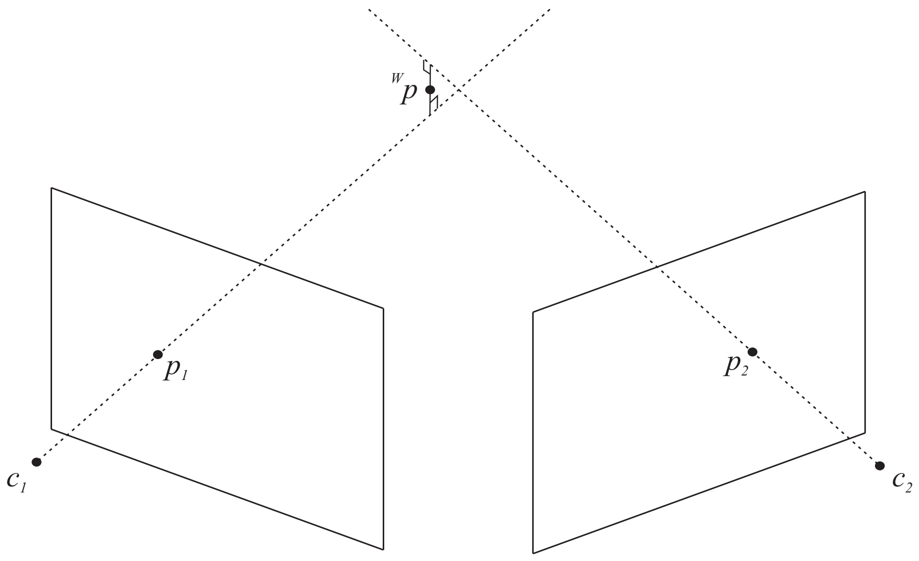


Figure 2.7: Stereo triangulation procedure. In this case, since the two lines-of-sight do not intersect, the final reconstructed point ^{W}p is computed as the midpoint of the smallest segment joining the rays (i.e., the one orthogonal to both).

where Π_x is the projection matrix of view x, and p_x the projection of ^{W}p in the camera x. Consequently, this leads to each view generating three equations constraining the system:

$$\begin{aligned} u(\pi_3^T \cdot {}^{W}p) - (\pi_1^T \cdot {}^{W}p) &= 0 \\ v(\pi_3^T \cdot {}^{W}p) - (\pi_2^T \cdot {}^{W}p) &= 0 \\ u(\pi_2^T \cdot {}^{W}p) - v(\pi_1^T \cdot {}^{W}p) &= 0, \end{aligned} \tag{2.12}$$

where π_j denotes the row j of matrix Π, and recall that a 2D point $p = (u, v)$.

However, we often deal with a more open problem, where camera positions are unknown, and have to be estimated together with the structure. It has been pointed out in the last section that the motion and the shape of the object are intrinsically related, and the Structure-from-Motion (SFM) problem tries to find both unknowns at the same time. The main advantages of SFM approaches is that they allow the use of a single camera, thus providing a flexible and accessible image acquisition system.

Starting from the corresponding points between images, these methods extract the camera's movement as well as a 3D reconstruction of those same points. In order to compute the motion the points describe, robust estimation methods like the ones presented in 2.4.2.2 are used. Once the motion is obtained, the 3D positions of the points are computed by triangulation. Note however, that having no other clue than image matches makes the results to be up to an unknown scale factor. This scale ambiguity, which is depicted in Figure 2.8, can be resolved to a posteriori by the use of additional information, such as knowing the size of any object in the scene.

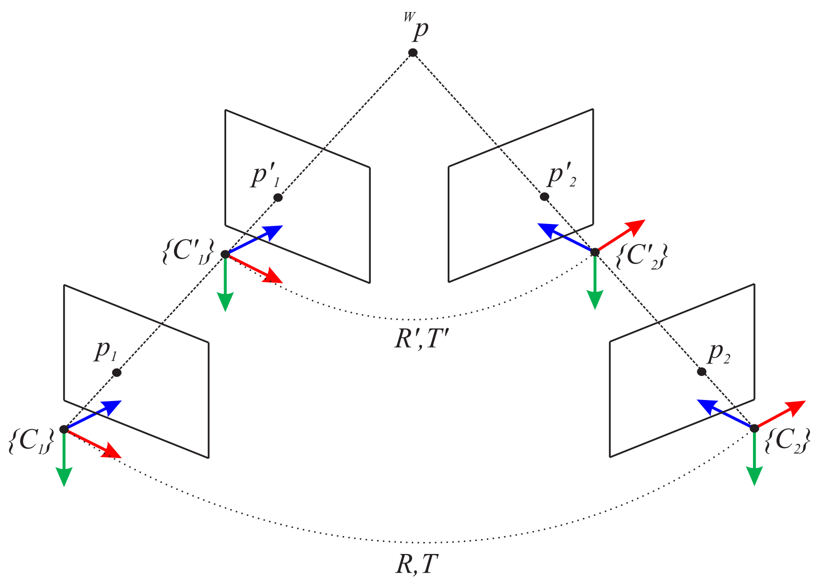


Figure 2.8: Scale ambiguity in SFM approaches. The same 3D point can be generated indistinctly by either C_1-C_2 or C_1'-C_2', where the two stereo systems differ by a scale factor.

The relative pair-wise motion between frames has to be accumulated in order to get the global motion of each camera with respect to the same reference frame. Obviously, the error accumulates quickly when adding the relative estimations. This is normally alleviated by the use of a Bundle Adjustment (BA) method, like that described in Lourakis and Argyros [146], which minimizes the global reprojection error.

Early approaches were based on applying the Fundamental matrix F (Beardsley et al. [21]) or the trifocal tensor (Fitzgibbon and Zisserman [80]) to compute relative motion between frames. Even in conjunction with BA, these methods suffer the problem of the F matrix being ill-conditioned to small motions, promoting a poor motion estimation in these cases (see Hartley and Zisserman [103]). A more general view is to recover the camera's pose directly by associating 3D points to its 2D projections in new images. Having this 3D-2D relationship, the DLT [103] is used as the model for a robust estimation method to compute the camera pose. Some examples of these techniques can be found in Brown and Lowe [39], Snavely et al. [197] and Nicosevici et al. [162].

It is worth noticing that the method we use to create the datasets presented in this thesis is a modification of the method in Nicosevici et al. [162]. In the original method, a sequential approach was presented. In our case, the sequential nature has been modified to follow the *unordered approach*, that is, we do not assume the images to come in a sequence, in this way making the method more generic. The pipeline followed in the method, closer now to the one shown by Snavely et al. [197], can be seen in Figure 2.9.

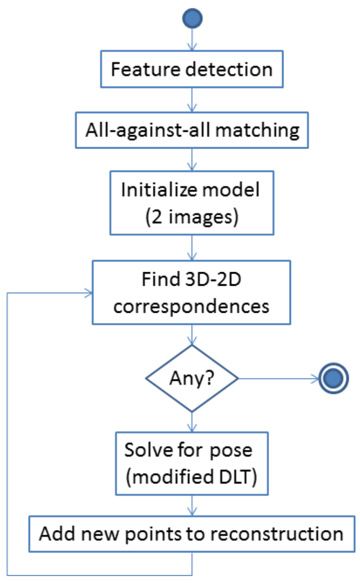

Feature detection
All-against-all matching
Initialize model (2 images)
Find 3D-2D correspondences
Any?
Solve for pose (modified DLT)
Add new points to reconstruction

Figure 2.9: Pipeline followed by the incremental Structure-from-Motion approach used in this thesis.

2.6 Dense Reconstruction

Dense reconstruction methods aim at finding the correspondence between points, as done in a previous step, but trying to maximize the number of matches. This is done by relaxing the distinctiveness requirement. In the previous correspondence search, we aimed at finding strong points that could be correctly matched across images, since they would be used to recover the motion parameters between frames, and their correctness would guide the accuracy of further steps. On the contrary, at this point we already have the cameras positioned, and we want to recover as many points as possible representing the object. Thus, not just feature points have to be recovered, but all the points/pixels in the image need to be candidates for matching. When reconstructing the surface, the main aim of this thesis, we need a consistent amount of points describing it, as having more samples of the object helps in disambiguating this ill-posed problem.

Several techniques have been proposed in this field, and relevant reviews and evaluations have been presented by Seitz et al. in [188] and Strecha et al. in [201]. However, there are two types of approaches that have proven valid in various scenarios, thus becoming popular with the research community.

The first, shown schematically in Figure 2.10, builds a depth map for each image using two or more images to validate the matches. Once the depth map is constructed for each image, and given the known poses of the cameras, these are registered into a common reference frame to provide a global representation of the object as a dense point set.

Obviously, this intensive matching approach is very likely to generate a huge number of outlier points, given that wrong or poor matches translate into wrong 3D locations that corrupt the final point set. Redundancy between different depth maps helps in eliminating some of these wrong points, but not all of them. Furthermore, some regularization steps may be used at image level to enforce coherence between neighboring matches (e.g., if the scene is supposed to be smooth, abrupt transitions between neighboring matches should be penalized). Despite the defects in the resulting point sets, these methods recover lots of 3D points, resulting in a dense sampling rate.

A popular approach in this direction is the *plane sweeping* method. Given a fixed reference view, and a set of compatible views, a set of candidate depths are tested for similarity. Thus, for each pixel in the image, there is a set of candidate 3D points (and consequently matches) that are scored by the similarity measure. The problem is then to select the best one. Naive strategies pick the best one as that obtaining a better similarity score, while others also include some regularity constraints and cast the problem as an energy minimization. Given each pixel being tested for correspondences in a fine-grained range of depth values, the method is mostly a brute-force approach. Nevertheless, its popularity has grown over the past few years for its ease of implementation with programmable graphics hardware. Proposals like Yang and Pollefeys [209] allow the execution of the many comparisons required to be performed in parallel with a common Graphics Processing Unit (GPU).

The other principal methodology consists of incrementally building the 3D point set using a greedy process. This type of methods starts with a set of seed points, and iteratively generates new points in the vicinity of the existing ones. In this case, the points are seen as small patches in 3D with a given orientation. The position and orientation of these patches is computed by optimizing their photoconsistency across images. In this way, the object grows incrementally, by making local decisions on which parts to generate next. This is opposed to the behaviour of processes based on depth maps, since the spatial coherence between them is not taken into account. This causes some parts to be overrepresented, as many depth maps may contribute to the same area. Also, because of this lack of coherence checks between depth maps, the resulting point sets provided by greedy methods tend to be smoother representations of the object. However, it is also worth noting that the number of points reconstructed is smaller in the greedy case. Two good examples of this category are Habbecke and Kobbelt [100] and Furukawa and Ponce [84], the last one being very popular because of its available open source implementation. A representation of the procedure followed in Furukawa and Ponce [84] can be found in Figure 2.11.

It is worth noting that this is the starting point of the present thesis. The lack of continuity in the most likely corrupted dense point set retrieved by the techniques presented above burdens further computations and proper visualization performed using this

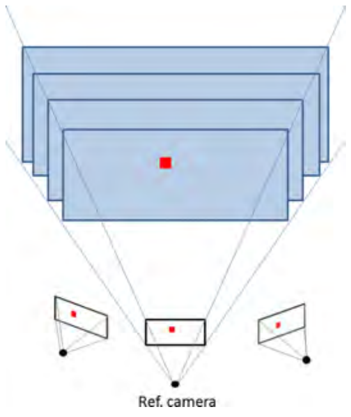
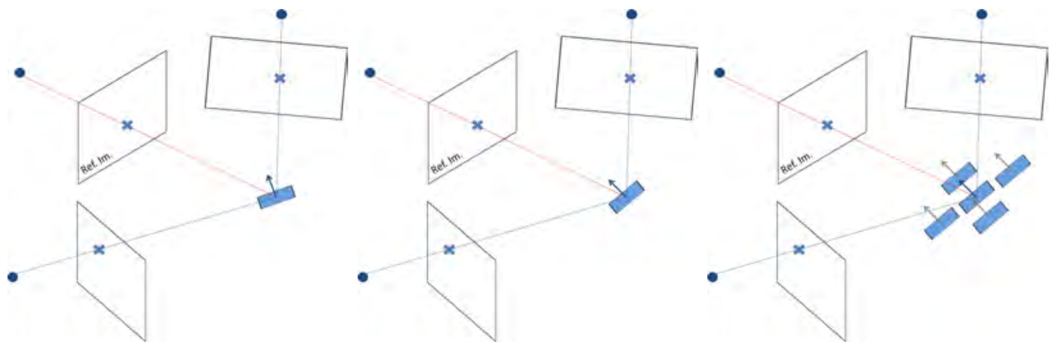


Figure 2.10: Depth map construction. Several candidate depths (blue rectangles) are tested, while each possible patch (red square) is given a similarity score based on its projection onto neighboring images.



(a) Initialization (b) Optimization (c) Expansion

Figure 2.11: Greedy dense reconstruction process. At first (a), the initialization starts by triangulating the feature points. Then (b), a 3D patch position and orientation is optimized for photoconsistency across images. Finally (c), new patches are generated near the initial point on the tangent plane defined by the patch. New points/patches are optimized again and steps (b) and (c) are repeated iteratively, in this way, growing the points which form the reconstruction.

data. Thus, we aim at obtaining a smooth continuous surface representation in the form of a triangle mesh. This mesh fills the gap in the optical object modelling pipeline, as it provides a virtual model suitable for processing using some of the mesh-based techniques available in the computer graphics or computational geometry literature. While being beyond the scope of this thesis, we present some of these methods in the next section for completeness.

2.7 Post-processing

Due to the ability of GPUs to render triangle primitives, this representation has been the most used to describe 3D objects since the beginning of computer graphics. After the recovery of the surface in the form of a mesh of triangles, one can benefit from the vast amount of available literature in mesh processing (see [34] for a broad overview of these techniques). Some of the processing that can be applied include:

- Simplification: Changes the complexity of the mesh. Real-time visualization of complex meshes often require the object to be represented at a resolution proportional to the position of the viewer: objects farther away from the viewer need fewer triangles to be perceived than closer objects. Multi-resolution structures can be built using simplification techniques in order to be able to dynamically change resolution depending on the needs.

- (Re)Meshing: Changes the quality/shape of triangles. Some computations, like the Finite Element Method (FEM), require the shape of the triangles to be close to regular, or adaptive to the complexity or curvature of the object. Meshing (or Remeshing) methods allow tuning the quality of the triangulation according to some user-defined parametrization on the shape of the triangles.

- Smoothing: Alleviates the roughness of the surface. Smoothing techniques can be applied to achieve a more visually pleasant representation of the object in areas of small high-frequency changes produced by errors in either the registering, measurement or reconstruction processes.

However, given the photogrammetric origin of the models generated, and in order to provide more realism to the models, the texture mapping approach is a post processing of great interest for modelling applications. In fact, given the known 3D positions of the cameras, the texture mapping process is quite straightforward: as shown in Figure 2.12, each triangle is projected in one of its compatible views, and its texture is extracted from there. The problem resides in selecting the most appropriate image to extract the texture

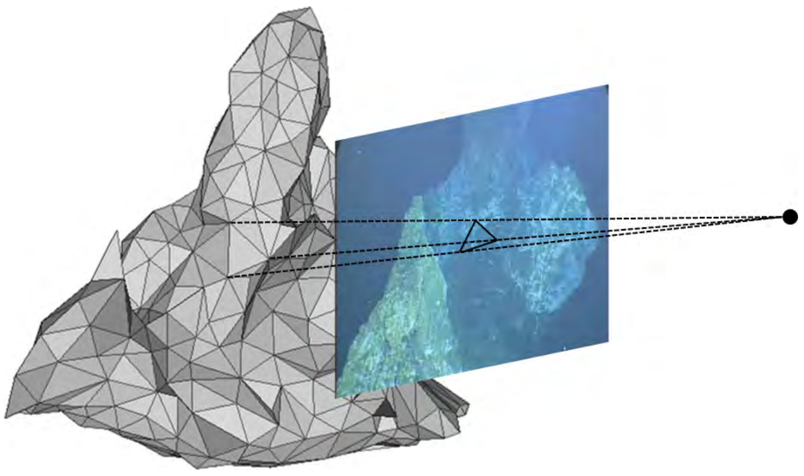


Figure 2.12: Texture mapping process, the texture filling a triangle in the 3D model is extracted from the original images.

from. This is done using heuristics that depend on the distance, since closer views of the object contain more texture information, and the orthogonality between the direction of the view and the normal of the triangle, since more orthogonal views provide a texture less influenced by projective deformations. However, blending methods might be needed to alleviate the differences in illumination on parts of the texture obtained from different images, as well as to alleviate possible camera registration errors. Some relevant methods in the literature are presented by Lempitsky and Ivanov [136] and Gal et al. [85].

2.8 Underwater 3D Modelling

In this chapter we reviewed a common 3D modelling pipeline. However, keep in mind that the present thesis is concerned with the problem of 3D underwater mapping. It is thus also important to know how the problem of 3D mapping has been tackled in the latter years in this hostile environment. In this section we review the methodology used nowadays in marine mapping, with a wider view that does not just include optical sensors, but other common techniques as well.

As pointed out in Chapter 1, underwater mapping relies primarily on assuming an underlying planar structure, common throughout the survey, where a 2.5D representation of the shape can be built. The problem of large area mapping is often tackled by using downward-looking cameras/sensors, since their overview capabilities provide an overall notion of the shape of the scene. The use of this approximation has motivated the wide spread use of scanning sensors located on the bottom of underwater vehicles, with their optical axis orthogonal to the seafloor.

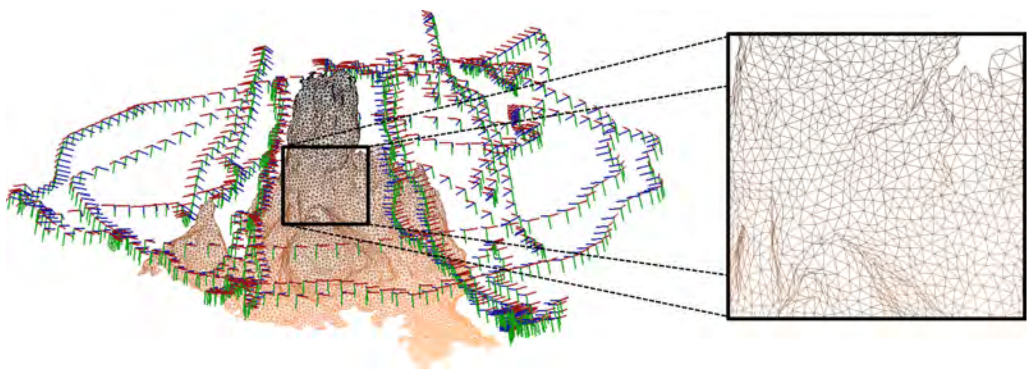
There is a great deal of literature on the different uses of this scanning configuration for sonar mapping, and automatic methods to build these maps out of raw sonar readings has been a hot research topic in recent years [174, 17]. The application of these techniques has proven to provide key benefits for other research areas such as archeology [25] or geology [210], to name a couple. Given the depth readings, retrieving a surface representation is straightforward. By defining a common plane, all the measures can be projected in 2D. Then, an irregular triangulation like Delaunay or a gridding technique can be applied to the projections on the plane and the 3rd coordinate is used to lift the surface in a 2.5D representation. These mappings can be enhanced by using photomosaics for texture, thus producing a multimodal representation of the area [114, 42].

On the other hand, although not that extensive, cameras are also used for heightmap reconstruction. Regarding full 3D reconstruction, the methods in the literature are more concerned with recovering the point set, obviating the surface reconstruction part. Thus, the focus is normally put on the registration of multiple camera positions, using either structure-from-motion systems [162, 40] or Simultaneous Localization and Mapping (SLAM) approaches [115] as well as monocular or stereo cameras [148]. The processing pipeline of these methods closely follows the one presented in the previous sections of this chapter. However, the application of these methods in the underwater domain has to face the added complexity of dealing with noisy images, in the sense of the previously commented aberrations introduced by the rapid attenuation of light on the water medium, the non-uniform lighting on the image, and forward/backward scattering. This translates in the methods requiring further preprocessing of the images to alleviate these issues, and having to deal with larger errors in the reconstructions than their less noisier on-land counterparts. Furthermore, as previously stated, the common configuration of downward-looking cameras makes methods tend to represent the shape underlying these points as a heightmap in a common reference plane, which may be easily extracted using PCA [194, 162].

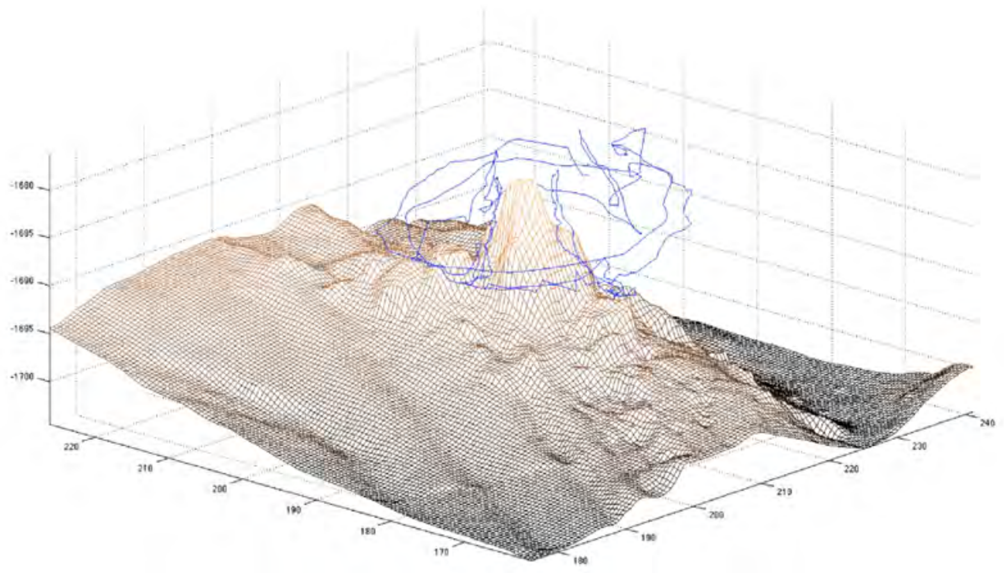
There are very few proposals on underwater mapping using some of the existing surface reconstruction techniques. In Johnson-Roberson et al. [115], the authors do use a volumetric method from Curless and Levoy [60], originally devised to full 3D reconstruction. Nevertheless, the camera is still observing the scene in a downward-looking configuration, and the added value of applying this method in front of a 2.5D approximation is not discussed. Nonetheless, they establish an approach to parallelize the reconstruction using this method on small chunks of data at a time and then merging the results. Another proposal, this time using a forward-looking camera and working on a more complex structure, is that presented in Garcia et al. [87], where an underwater hydrothermal vent is reconstructed using dense 3D point cloud retrieval techniques and the Poisson surface reconstruction method [118].

The review of the state of the art on underwater 3D modelling has revealed that most of the proposals retrieve the surface in a 2.5D representation. Furthermore, given the straightforwardness of changing from depth readings to this representation, the creation of a triangulated elevation map is usually considered a side result. However, in this thesis we go for a more general scenario, where the sensor can be mounted in a general configuration; that is, located anywhere on the robot and with any orientation. With this new configuration, objects can be observed from new viewpoints, so that the retrieved ranges are no longer suitable for projection onto a plane (and hence, a 2.5D map cannot be built). Viewing the object from arbitrary positions allows a better understanding of the global shape of the object since its features can be observed from angles that are more suitable to their exploration. Take as example the trajectory in Figure 2.13, corresponding to a survey of an underwater hydrothermal vent (further results with this dataset are illustrated in Section 5.5). It is impossible to recover the intricate structure of this area with just a downward-looking camera. In this case, the camera was located on the front of the robot looking approximately at a 45^{o} angle. This configuration allowed the recovery of the many concavities this object contains. The difference in resolution and complexity of this same area when mapped using multibeam sonar technologies in a 2.5D representation are evident.

In comparison with current approaches for 3D mapping, we aim to deal directly with the problem of surface reconstruction by giving it the relevance it deserves. The contribution of the present thesis is to be able to represent complex 3D structures that are not representable using heightmap representations, while also providing noise correction and outlier removal.



(a)



(b)

Figure 2.13: (a) Example of a more general trajectory, with a frame representing each camera (Red/Green/Blue corresponding to the X/Y/Z axis). This is the trajectory used for mapping the Tour Eiffel hydrothermal chimney at a depth of about 1700 meters in the Mid-Atlantic ridge. In this case, the camera was pointing towards the object in a forward-looking configuration. The shape of the object shown was recovered using our approach presented in Chapter 5. Note the difference in the level of detail when compared with a 2.5D representation of the same area obtained using a multibeam sonar in (b). The trajectory followed in (b) was downward-looking, hovering over the object. The trajectory in (b), shown in blue, corresponds to the survey in (a).

Chapter 3

Related Concepts

3.1 Introduction

In this chapter, we present some useful concepts required to understand the basis of the surface reconstruction methods that will be presented in the literature review in Chapter 4, as well as our further proposals in Chapters 5, 6 and 7.

We start by giving the definition of a surface, as well as reviewing some of its main types of representations and their properties. Then, we briefly introduce some common algorithms and data structures in computational geometry, as these are extensively used in the literature. Finally, we focus on the specific problem of surface meshing. Given the problem at hand, and when not stated otherwise, we assume to be working on $\mathbb{R}^3$ throughout this chapter, despite that many of the following definitions may be generalized to higher/lower dimensions.

3.2 Surfaces

Before dealing with the surface reconstruction problem, it is important to start by describing what a surface is. Surfaces (as well as curves) are a type of topological spaces called manifolds. More precisely, a topological space consists of a point set P associated with a series of neighborhood relationships. These neighborhoods are defined as a set of open subsets T that satisfy the following conditions:

- The empty set $\emptyset$ is contained in T.
- P is in T.
- The union of an arbitrary number of sets is also in T.
- The intersection of a finite number of sets is also in T.

Regarding the definition of a manifold, another important concept to define is that of homeomorphism. A homeomorphism is a map $h : X \to Y$ from space X to space Y which is bijective and has a continuous inverse. This map defines an equivalence relationship between two topological spaces. Broadly speaking, and taking the specific example of two surfaces, they are homeomorphic if one can be transformed to the other by twisting, squeezing, or stretching without cutting or attaching.

Given the concepts of topological space and homeomorphism, we can now define a k-manifold as a topological space that is locally homeomorphic to the Euclidean $\mathbb{R}^k$. More precisely, surfaces are 2-manifolds, where around each point its neighborhood is homeomorphic to the open 2-ball, i.e., an open disk. Since any open k-ball is homeomorphic to $\mathbb{R}^k$, this implies that, on a small scale around a point in the manifold, the neighborhood relations resemble that of $\mathbb{R}^2$ (i.e., the plane).

In this section, we further detail the most commonly used representations to describe a surface, and a possible classification of surfaces based on some properties.

3.2.1 Surface Representation

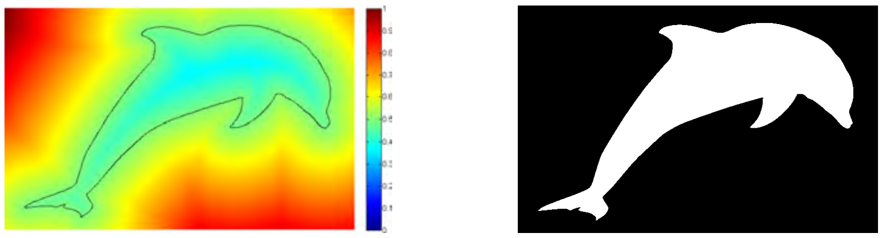
There are mainly two types of representations for a surface: explicit and implicit. On the one hand, the explicit formulation, also called parametric, consists of defining the 3D surface as a function depending on 2D parameters. On the other hand, the implicit formulation defines the surface as an isovalue inside a scalar field. These formulations are explained in detail below.

3.2.1.1 Implicit

The volumetric view of the implicit formulation casts the surface as a level set inside a scalar valued function $I : \mathbb{R}^3 \to \mathbb{R}$, so that $S = \{p_x \in \mathbb{R}^3 | I(p_x) = k\}$, where k is the iso-level defining the surface. Without loss of generality, and following the majority of approaches, the zero level set $k = 0$ is considered to describe the surface.

Some examples of implicit definitions include radial basis functions [43], algebraic surfaces, multilevel partition of unity [165], or some moving least squares surfaces [5]. These representations usually come in the form of distance functions, i.e., a point $p_x \in \mathbb{R}^3$, $I(p_x) = dist(p_x, S)$. However, in some cases, cruder representations like an inside/outside labelling may be sufficient to represent a closed surface. In this latter case, the function is called the indicator function χ of the surface, which, when given a point, the function is 0 if the point is located outside the object O, and 1 if located inside, i.e.:

$$\chi(p_x) = \begin{cases} 1 & \text{if } p_x \in O \\ 0 & \text{if } p_x \notin O. \end{cases} \tag{3.1}$$



(a) Implicit function (b) Indicator function

Figure 3.1: Implicit/Indicator functions in 2D. The value for each point in the implicit function in (a) corresponds to the distance from a given point to the curve defining the object, while the indicator function in (b) provides an inside/outside labelling for each point with respect to the shape.

Obviously, in the case of indicator functions, the surface S is found at the interphase between 0 and 1 labels. A 2D example of both methodologies can be seen in Figure 3.1.

In order to apply easy computations to implicit representations, the continuous implicit function is discretized in some bounding box around the object. Arbitrary evaluations of the functions inside the discretized domain are then obtained through interpolation.

A simple and common approach is to divide the embedding space into a regular voxel grid. This basic subdivision of the embedding space provides the advantage of being very easy to manipulate. However, the memory consumption costs increase dramatically with the accuracy, as it is directly related to the size of the voxels (e.g., halving the size of a voxel, and consequently doubling precision, increases the grid size cubically).

In order to alleviate memory issues, some adaptive data structures have been proposed in the latter years. These data structures consist of faithfully representing the implicit function at the interest areas (i.e., near the surface), and representing naively non-interesting parts (i.e., far from the surface). Hierarchical octrees [207] or tetrahedral meshes [8] are examples of these representations.

Implicit surfaces are well suited to inside/outside geometric queries, useful in Constructive Solid Geometry (CSG). Furthermore, the evolution of the surface is easily tracked using an implicit formulation. However, it is difficult to sample the surface exactly using this representation, as is moving a point *onto* the surface.

As previously stated, even in the case of the surface being implicitly defined, our goal is to extract its explicit form as a triangle mesh. In Section 3.4, we discuss the common methodology for conversion between these two representations.

3.2.1.2 Explicit

An explicit surface consists of a function $f : \Sigma \to S$ mapping the $\Sigma \in \mathbb{R}^2$ parameter domain to the surface $S = f(\Sigma) \in \mathbb{R}^3$. Simple shapes have an explicit formulation that is defined over a range of parameter values. For example, a sphere may be defined using spherical coordinates as follows:

$$\begin{aligned} x &= r\cos\theta\sin\phi \\ y &= r\sin\theta\sin\phi \ , \\ z &= r\cos\phi \end{aligned} \tag{3.2}$$

where r is the radius of the sphere, and the parameters θ and ϕ range from 0 to 2π and from 0 to π, respectively. However, a simple parametric formulation like the one above may be nonexistent to represent more complex surfaces accurately enough. For this reason, a more general way to represent surfaces is to define them in piecewise parametric form. A piecewise parametric representation divides the surface into small parametric parts, called surface patches, which define the global surface when joined together. Obviously, the size and number of surface patches defining the surface rule its accuracy (see Figure 3.2).

The most common piecewise parametric surface representations are triangle meshes. A triangle mesh can be represented as a graph structure, having a set of vertices $V = v_1, ..., v_n$, with associated points $p(v_i)$ defining its 3D position. Besides, neighborhood relationships between vertices are defined by a set of faces connecting them $F = f_1, ..., f_n$ ($f_i \in V \times V \times V$) and/or a set of edges $E = e_1, ..., e_n$, $e_i \in V \times V$. On top of that, 2D positions can also be associated to each of the vertices in the mesh, so that the $\mathbb{R}^2 \to \mathbb{R}^3$ parametrization can be directly defined using the barycentric coordinates of each triangle. Barycentric coordinates define a linear 2D patch using each triangle $t = [p_a, p_b, p_c]$ via the following parametric representation:

$$p = \alpha p_a + \beta p_b + \gamma p_c, \tag{3.3}$$

where $\alpha + \beta + \gamma = 1$ and $\alpha, \beta, \gamma > 0$.

There are many advantages when using this representation. On the one hand, it is suitable for visualization with common graphics hardware. Furthermore, having a parametric (or piecewise parametric) representation allows the reduction of the 3D complexity of some problems to its 2D counterpart in the parameter domain. For these reasons, problems such as sampling the surface are easily accomplished, as they only require a sampling in the space of the parameters. On the other hand, there are also some disadvantages in using this representation. An obvious one is that manipulation of this surface is not trivial, since a change in the shape of the surface also implies the need to update the parametrization. All the same, the problem of tracking an evolving surface requires

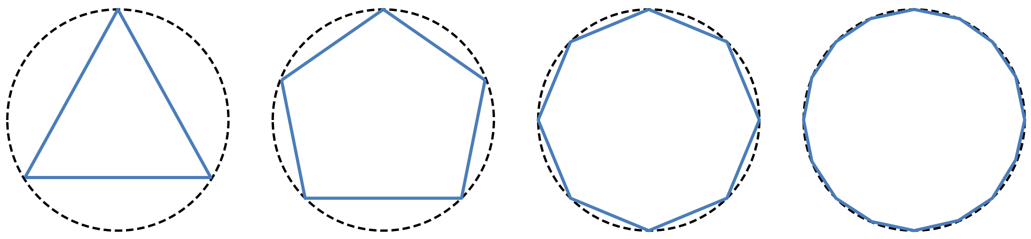


Figure 3.2: Piecewise-explicit representations of a circle, with increasing resolution from left to right.

tracking and modifying the topology and parametrization of the surface. By comparison with the properties of the implicit formulation discussed in the previous section, we can conclude that the advantages and drawbacks of the explicit and implicit representations are complementary. Thus, the application of one representation or the other is application dependant.

3.2.1.3 Representation Selection

The aim of this thesis is to develop surface reconstruction methods for seafloor modelling purposes. Thus, the applications of the results we obtain are mainly visualization (enhaced by texture mapping), and light manipulation, such as point-picking or area/volume computations. Based on these post-processing purposes, and checking the above mentioned advantages and drawbacks for each type, in this thesis we require the methods to finally provide the surface in a piecewise explicit representation. Note that this requirement does not ban the use of methods working in an implicit formulation, since a surface in this representation can converted to its explicit form using one of the methods in Section 3.4. Following the vast majority of the literature, we refer hereafter to the piecewise parametric representation with triangle surface patches as simply the explicit representation.

Despite this choice, there are other valuable and commonly used representations of surfaces that we refrain from further commenting on as they fall beyond the scope of this thesis. Some examples include quadrangle meshes (where the surface patch is a quadrangle) [33], spline surfaces [77, 172], and subdivision surfaces [219].

3.2.2 Surface Classification

A given surface (or its explicit/implicit representation) can be broadly classified according to some properties. In our context, the most important are **smoothness**, **manifoldness**, **orientability**, and the existence of **boundaries**. Figure 3.3 exemplifies this classification according to these characteristics. Surface reconstruction algorithms differ in the assumptions they make on some of these properties, as well as their behaviour with respect to

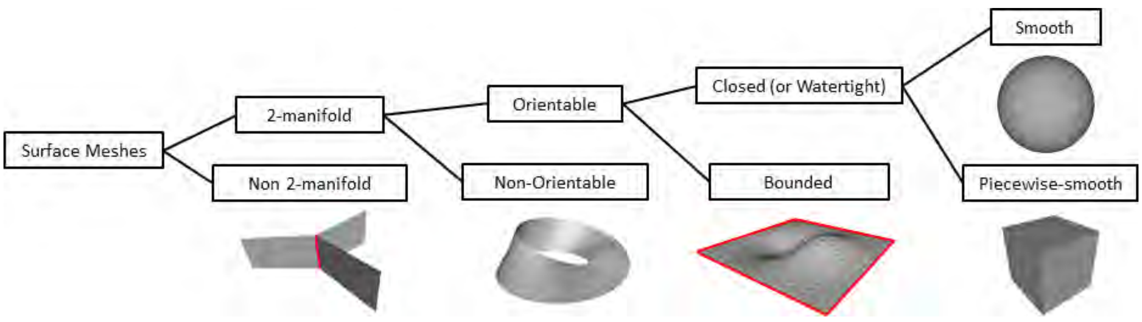


Figure 3.3: A sample of surfaces classified using the criteria described in section 3.2.2.

them. We detail them more precisely below.

Smoothness A surface S is smooth if it is differentiable everywhere, i.e., it has well defined derivatives at every point $p \in S$. Roughly speaking, taking an arbitrary point p from a smooth surface, at a very small scale the function near this point resembles a plane. In this thesis, we assume the surfaces to reconstruct to be smooth. Other important types of surfaces are the piecewise-smooth ones, consisting of smooth patches joined together through sharp curves or corners.

Manifoldness Recalling the definition of a surface in Section 3.2, a 2-manifold surface is that having its neighborhood relations homeomorphic to a topological disk. The problem of non-manifoldness is related to the problems during the generation of the representation of this surface. Triangle meshes may contain some degeneracies that prevent the mesh from being manifold. For an explicit surface to be manifold, it requires having every vertex surrounded by a *fan* of faces, and also that any given edge has at most two incident faces. In cases where these requirements are not fulfilled, the mesh is non-manifold. Very few methods in the mesh processing literature allow for a non-manifold input, promoting the use of mesh repairing techniques [141, 163, 26].

Orientability The orientation of a face is a cyclic order of its vertices. Two adjacent faces are compatible when the vertices of their common edge are in opposite order. A manifold mesh is orientable if any two adjacent faces have compatible orientation. Some surfaces, like the Klein bottle or the Möbius strip, are unorientable, i.e., a consistent orientation cannot be devised.

Boundaries There exist boundaries in a manifold mesh when edges have just one incident face. It is often assumed in many papers to work with manifold surfaces without boundaries, also referred to as closed or watertight. However, as pointed out in previous chapters, we do not make this assumption.

3.3 Computational Geometry

The computational geometry research field has played an important role in the area of surface reconstruction. More specifically, the Delaunay and Voronoi structures are the base of many approaches in the state of the art, as well as the methods proposed in this thesis. The base space decomposition proposed by Delaunay or Voronoi, along with their many properties, allow the definition of surface reconstruction methods with theoretical guaranties, as will be seen in Chapter 4. In the context of this thesis, the Delaunay triangulation is the base of the surface mesher method that will be presented in Section 3.4.2, which is used in the methods proposed in Chapters 5, 6 and 7. Furthermore, we also use the Delaunay triangulation to store irregular implicit functions in Chapter 7. In order to fully understand the insights of these types of approaches, we present in the following, some useful definitions from the field.

Throughout this section, we assume a general position on the point sets. This assumption is often taken in the computation of many geometrical structures in computational geometry (such as the Delaunay triangulation or the Voronoi diagram). A set of points in $\mathbb{R}^3$ is in a general position if there are no degeneracies such as having three collinear points, four points in a common circle/hyperplane or five points on a common sphere. Even in the case of P not being in general position, algorithms can also apply the Simulation of Simplicity (SoS) method [73] in order to eliminate the special cases rising from degeneracies. These techniques are used in order to avoid dealing with special cases when defining the algorithms.

3.3.1 Simplicial Complex

When talking about triangulations, one often refers to the usual case, where a surface or a plane is subdivided into a set of triangles. However, the triangulation concept can be generalized to an arbitrary number of dimensions, and adopt further configurations.

A **k-simplex** is the generalization of a triangle for a dimension k. It is defined as the convex hull spanned by $k + 1$ vertices. Thus, as illustrated in Figure 3.4, in an $\mathbb{R}^3$ space, one can find 0-simplices (vertices), 1-simplices (edges), 2-simplices (triangles) and 3-simplices (tetrahedra). At the same time, a k-simplex is formed by faces of dimension $d < k$. The boundary of a k-simplex has $k + 1$ 0-faces, $k(k + 1)/2$ 1-faces and $\binom{k+1}{d+1}$ d-faces in general. If we take as example the 2-simplex (a triangle), it contains three 1-faces (edges) and three 0-faces (vertices).

Given the definition of simplices, **simplicial complexes** are used to generalize triangulations to arbitrary dimensions. A simplicial complex K is a finite set of simplices, so that all the faces forming its simplices are in K and the intersection of any two sim-

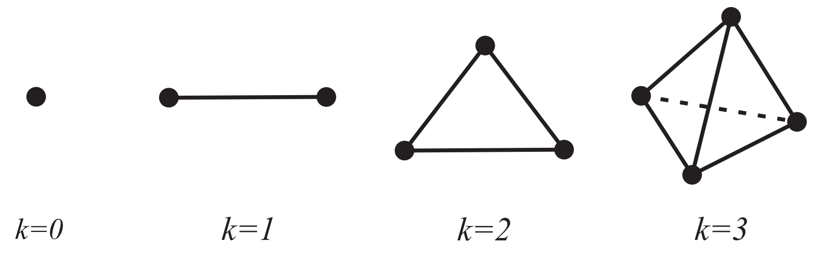


Figure 3.4: Examples of a k-simplex for k ranging from 0 to 3.

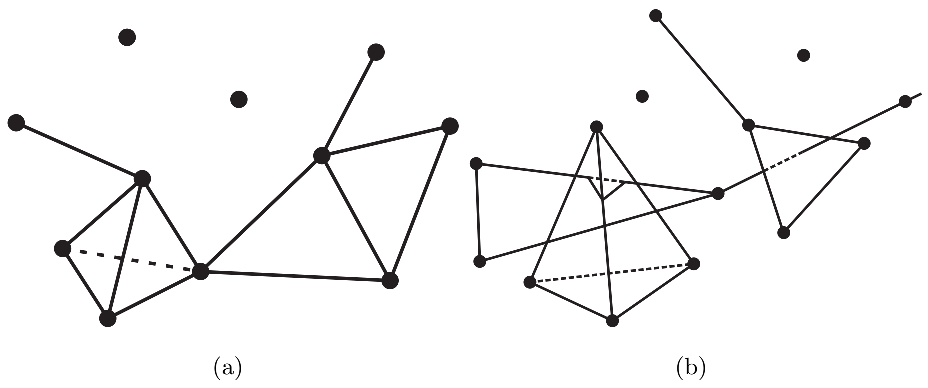


Figure 3.5: A simplicial complex is depicted in (a). Note that (b) is not a simplicial complex, as some of its simplices intersect without sharing a face.

plices is already a face of one of them. This definition raises complexities like the one in Figure 3.5(a), but forbids configurations like the one on Figure 3.5(b). Furthermore, the dimension of K is defined by the maximum of the dimensions of the simplices conforming it.

While more restrictive, notice that the previously defined explicit surfaces representation also describes simplicial complexes.

3.3.2 Convex Hull

The convex hull $CH(P)$ of a set of points P is the intersection of all the convex sets containing P. Linking with the previous definition of simplicial complexes, it can also be referred to as the union of all the possible simplices that can be generated by P. Also, in an informal way of speaking, the $CH(P)$ can be seen as the shape the set of point forms when covered by an elastic material, like a rubber band.

3.3.3 Delaunay Triangulation

Given a set of points P, the Delaunay triangulation Del(P) is a simplicial complex that breaks down the convex hull of P into tetrahedrons, so that no point in P is inside the circumsphere of any tetrahedra in Del(P). Introduced by Boris Delaunay in [61], one of the main advantages of this triangulation is its maximization of the minimum angles among all possible triangulations of a given vertex set. In other words, it produces a triangulation with triangles as close to regular as possible.

3.3.4 Voronoi Diagram

The Voronoi diagram Vor(P) [205] is a decomposition of $\mathbb{R}^3$ into a set of convex polyhedrons determined by distances between sets of points. Each point p in the set P defines a Voronoi cell V_{p_i}, which contains all the points in $\mathbb{R}^3$ that are closer to p_i than to any other point in P:

$$V_{p_i} = p_x \in \mathbb{R}^3 : \forall p_j \in P \, \|p_x - p_i\| < \|p_x - p_j\|. \tag{3.4}$$

3.3.5 Delaunay/Voronoi Duality

A relevant property between the Delaunay triangulation and the Voronoi diagram is that they are **dual**. That is, when the intersection between cells $V_{p_1}, V_{p_2}, ..., V_{p_n}$ in the Voronoi diagram is not empty, then CH($p_1, p_2, ..., p_n$) is a simplex of the Delaunay triangulation. Table 3.1 lists these dualities in $\mathbb{R}^3$.

3.3.6 Delaunay/Voronoi Related Concepts

In this section, we describe some of the concepts related to the Delaunay triangulation and the Voronoi diagram that are useful for the analysis of the computational geometry methods in surface reconstructions.

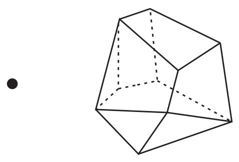
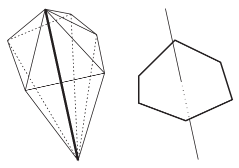
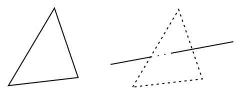
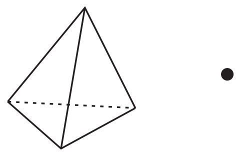
3.3.6.1 Poles

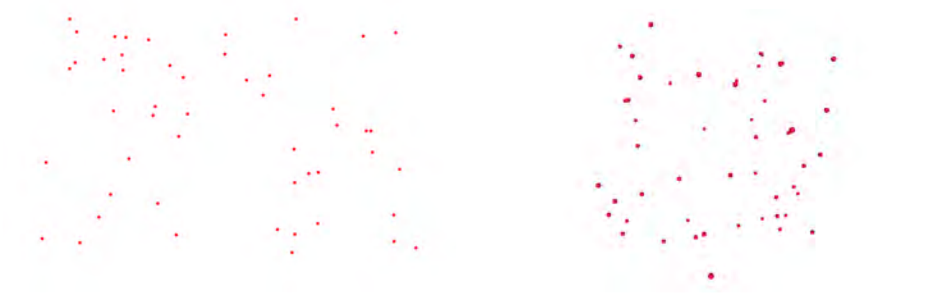
Given a Voronoi cell V_p, the poles are defined as two specific vertices. On the one hand, the positive pole q_p^+ is defined as the vertex of V_p farthest from p. As will be discussed in Chapter 4, the vector u_p^+ joining p and q_p^+ can be used as an approximation of the normal at p under some conditions. On the other hand, the negative pole q_p^- is defined as the point furthest from p, so that the vector u_p^- joining q_p^- and p makes an angle with u_p^+ higher than $\pi/2$.

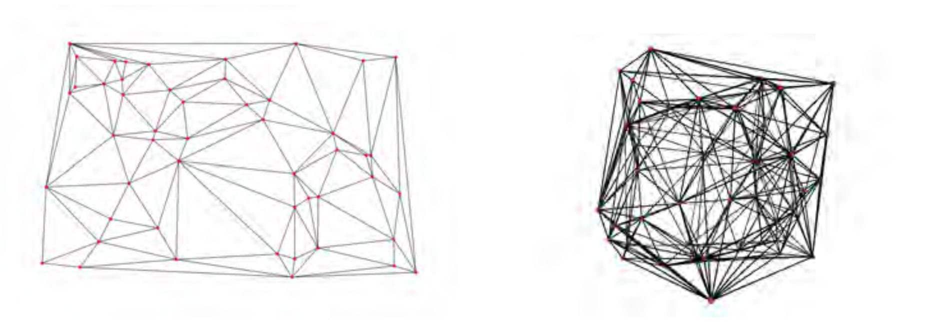
3.3.6.2 Delaunay Subcomplexes

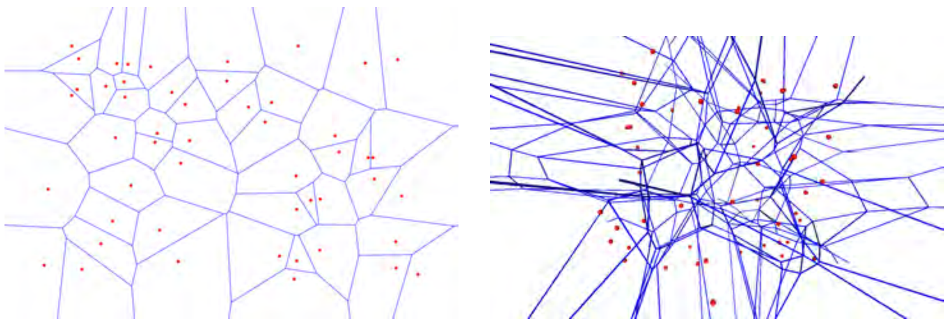
Using the Delaunay triangulation as a base, one may define a subcomplex by restricting it to a subset of its simplices following some properties. In this section, we define two of the most popularly used in the literature.

Table 3.1: List of dualities between the Delaunay triangulation and the Voronoi diagram in 3D.

Delaunay	**Voronoi**	**Depiction**
vertex	cell	
edge	face	
triangle	edge	
tetrahedron	vertex	


(a) Point sets


(b) Delaunay triangulation


(c) Voronoi diagram

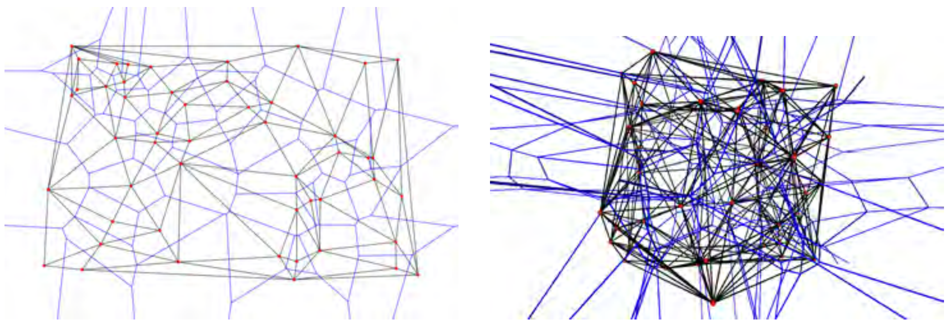

(d) Delaunay/Voronoi duality

Figure 3.6: Sample of the Delaunay/Voronoi decompositions, with its duality relationships (left column in 2D and right column in 3D). From first to last row, we find the input point set, its Delaunay triangulation, the corresponding Voronoi diagram, and both constructions together (in order to show their duality).

Gabriel Complex A Gabriel simplex is that having its circumscribing sphere empty of other points. Note that by definition, all tetrahedra inside the Delaunay triangulation fulfills the Gabriel property. However, this is not the case for the lower dimensional simplices (triangles and edges). Nevertheless, all the Gabriel simplices are contained in the Delaunay triangulation. Gabriel complexes, k- $\mathrm{GC}(P)$, can be defined by retaining all the Gabriel simplices of dimension k from the Delaunay triangulation.

α-Shape The α-shape definition provided by Edelsbrunner and Mucke[72] is one of the first trials in the literature to formalize the concept of *shape* defined by a point set P. α-shapes are generalizations of the convex hull of a point set guided by a parameter α, which are demonstrated to be contained in $\mathrm{Del}(P)$. This parameter ranges from $\alpha = \infty$, where the α-shape is equal to the convex hull of the points, to $\alpha = 0$, where the α-shape is equivalent to the point set itself. It is defined by the domain covered by the simplices in $\mathrm{Del}(P)$, whose empty circumscribing sphere has a radius $r \leq \alpha$. More intuitively, one can imagine having an eraser ball of radius α, which one can use to *erase* the space, but that can not remove the points, which are fixed and do not let the eraser pass through them. Starting with a *filled* space and using this spherical eraser, one can remove all the parts of the space into which this spherical eraser can fit. The space not reacheable by the eraser will result in a shape containing spherical caps and arcs. This generic definition is called the α-hull, and is related to the α-shape by substituting the spherical caps between triplets of points by triangles, and arcs by segments.

3.3.6.3 Constrained Delaunay

A constrained 2D Delaunay triangulation is that forced to contain a set of input edges as part of it. In the 3D case, not just edges but also planar faces may be required to be contained in the triangulation. While there is always a constrained triangulation in 2D, this is not the case in 3D, which may require the addition of Steiner points [192].

Even if semantically similar, it is important not to confuse this concept with that of Restricted Delaunay triangulations, defined in later chapters.

3.3.6.4 Power Diagram/Regular Triangulation

The power diagram and the regular triangulation are the generalizations of the Voronoi diagram and the Delaunay triangulation respectively for a set of weighted points. In this sense, each point p_i has an associated weight w_i. These weights give rise to a redefinition of the distance function from an arbitrary point p_x in space:

$$dist_{pow}(p_x, p_i) = \|p_x - p_i\| - w_i. \tag{3.5}$$

Thus, given the new distance function (referred to as the power distance), the power diagram is the decomposition of $\mathbb{R}^3$ into convex cells so that:

$$PD_{cell}(p_i) = \left\{p_x \in \mathbb{R}^3 : \forall p_q \in P, dist_{pow}(p_x, p_i) \leq dist_{pow}(p_x, p_q)\right\}. \tag{3.6}$$

As previously observed for Voronoi/Delaunay, the duality of the power diagram decomposition gives rise to the regular triangulation.

3.3.7 Medial Axis

The Medial Axis (MA) MA(S) of a surface S corresponds to the closure of points having more than one closest point on S. It provides a skeletal representation of the shape and is used in many applications, like shape recognition [27]. A closely related concept is that of the medial axis transform MAT(S). The two (or more) nearest neighbors on the boundary surface S define a set of maximal balls, i.e., empty balls not containing any points from S. The MAT(S) correspond to this collection of balls, whose centers are on MA(S). This transformation is bidirectional: given S, one can obtain MAT(S), and from MAT(S) one can recover S. This duality is exploited by some surface reconstruction algorithms.

3.3.7.1 Local Feature Size and ϵ-Sampling

The feature size $lfs(p)$ is defined as the minimum distance from a point p located on a surface S to the medial axis MA(S). Using this concept, Amenta and Bern [11], [10] presented a non-uniform sampling definition called the ϵ-sampling. Given an $\epsilon > 0$, a set of points is an ϵ-sampling of a surface S if every point x on S has a sample on P with a distance of at most $\epsilon \cdot lfs(x)$. Even if this sampling condition allows the definition of algorithms for surface reconstruction with theoretical guaranties, it is obvious that it cannot be checked as a step prior to reconstruction, as its definition requires the unknown surface S.

3.4 Surface Meshing

The meshing term is quite generic. In short, it consists of generating a mesh of a surface from an already known representation of that surface. Because of its large application in very different research fields, one of the most important problems is to extract triangle meshes from implicit representations. For this reason, we separate the methods devised to polygonize implicit surfaces from more generic ones, which can be applied to different kinds of representations. Note also how this problem is intrinsically related to surface reconstruction: if we are able to devise some representation of the surface of the object from a set of points on this surface, this representation can be meshed using the methods in this section in order to extract the required triangle mesh.

Note, however, that we intentionally omit the approaches that only accept as input an already existing triangulated mesh as the surface approximation. This problem, referred to as remeshing, has some specific approaches taking advantage of the triangulated input. Obviously, these types of methods have no interest in our scenario, since the triangulated surface is what we expect the surface reconstruction process to provide. Nonetheless, they might be suitable as a post-processing step once the surface has been recovered in order to enhance some of its properties. For further reference on this subject, the reader is referred to the extensive survey proposed by Alliez et al. [9].

The problem of surface meshing has been largely discussed at length over the last decades. The present definitions do not intend to be a thorough survey of the matter, but a review of the most commonly used algorithms and their variants. Finally, it is worth focusing on the meshing method of Boissonnat and Oudot [30], presented in the last section, as its modification is the basis of the surface reconstruction proposals presented in Chapters 5 and 6.

3.4.1 Isosurface Extraction

As we have seen in Section 3.2.1, explicit and implicit representations have complementary sets of advantages and disadvantages. Thus, the representation required depends on the application of the data. Consequently, the conversion between these two representations may be required.

The conversion from an explicit to an implicit representation of a surface is quite straightforward. Given a discretization of the space (e.g., a voxel grid), the value of an implicit surface can be constructed by computing the signed distance from each discrete position to its closest primitive on the surface. This way, the signed distance function represents the surface implicitly. This distance calculation may be computationally expensive, but data structures like k-D Trees [82] can be used to alleviate the cost, introducing fast closest distance queries.

On the contrary, the conversion from implicit to explicit representation has no direct formulation. The problem of finding an explicit representation, normally in the form of a triangle mesh, from an implicit one, is known by the name of *isosurface extraction* or, more generally, *isosurface meshing*. In this section, we present a review of the available methods used for this purpose.

3.4.1.1 Marching Cubes

Marching Cubes is the most popular and used isosurface extractor method. Created by Lorensen and Cline [145], it extracts a triangle mesh approximating an isosurface defined by a scalar field. Having the implicit surface discretized in a regular grid, the algorithm checks the scalar value at the 8 corners of each voxel to monitor if the surface passes though it or not. Taking into account that each vertex may contain a larger or smaller value than the one defining the required surface (the isovalue), the number of possible cases is restricted to $2^8 = 256$. However, by taking into account reflective and rotational symmetry, the authors show that these 256 cases can be summarized into 15 possible configurations. The final position of the vertex along an edge of the cube is a linear interpolation of the scalar values at the vertices of that edge.

Even though its initial implementation assumes the implicit domain to be discretized on a regular voxel grid, further improvements are devoted to using it on irregular data structures like octrees [120]. Also, some extensions on the method have been proposed in order to deal with sharp features, like edges or corners [123].

3.4.1.2 Dual Contouring

Quite related to the Marching Cubes approach, dual contouring methods [116] extract the isosurface from the scalar field in a dual mesh. The main advantage of dual approaches is their ability to represent sharp features on the mesh. However, it requires the input implicit surface to provide gradients or surface normals. When not available, some approximation has to be devised (e.g, they can be extracted using finite differences on the scalar function).

As in Marching Cubes, the process starts by computing implicit values at grid vertices, and tagging each vertex as being inside or outside the surface. Then, the voxels whose vertices have a mixture of inside/outside labels are the ones being intersected by the surface and, consequently, taken into account. The main difference with Marching Cubes comes at this point of the algorithm, since, instead of using a list of casuistics in order to resolve the shape of the intersection inside the voxel, the dual approach generates a vertex inside it. Thus, each intersecting voxel contains a new vertex, called the dual vertex. The final connectivity of the surface is generated by connecting neighboring dual vertices.

Obviously, the most important step of this method is how to generate the dual vertex at each voxel. Considering the normals at the intersection points in the edges of the voxel, the dual vertex location p_d is obtained by minimizing an error functional defined by the intersection of planes spanned by those normals:

$$E(p_d) = \sum_{i=1}^{n} ((p_d - p_i)n_i)^2, \tag{3.7}$$

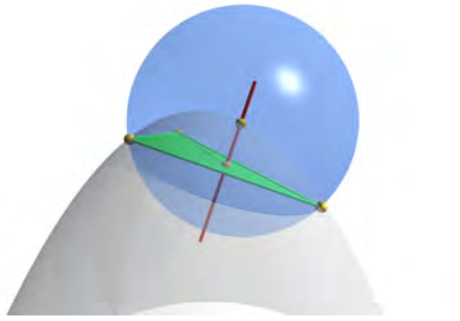


Figure 3.7: Example of a surface Delaunay ball.

where p_i is the edge intersection point i, and n_i its corresponding normal. This results in solving a linear system of equations. Note however that the obtained explicit mesh may contain not just triangle patches, but polygons with an arbitrary number of edges. Thus, these polygons have to be triangulated in order to obtain a triangle mesh.

Further improvements on the method refer to better preservation of manifoldness [182] or to improve its application on adaptive octrees [183] (note that the original method does not require regular grids).

3.4.2 Restricted Delaunay Meshing

There are cases where not only no explicit representation of the surface is needed, but also that the mesh is required to follow some characteristics. This may not be just an aesthetic problem (i.e., having *nicer* triangles), but could also be required for further processing applied to the mesh. This is the case, for instance, in Engineering computations like FEM, where having a regular discretization of the mesh provides more accurate results on the calculus involving this surface. In triangle meshes, the term regular refers to the shape of its triangles. The methods following the Restricted Delaunay paradigm allow the user to tune some characteristics of the final mesh. Consequently, these methods are also useful in the fields of data compression or mesh simplification, since the user-defined properties of the final mesh usually involve an approximation bound to be considered, which provides the user with control of the tradeoff between complexity and accuracy.

Given a set of points E on or near a surface, the Restricted Delaunay Triangulation (RDT) is a subcomplex of the 3D Delaunay triangulation of E formed by the Delaunay triangles whose dual Voronoi edges intersect the surface. Each RDT triangle has a circumball centered on the surface and empty of all other E points. This circumball is called a surface Delaunay ball (see Figure 3.7).

Boissonat and Oudot [30] proved that if the sampling E of the surface is dense enough with respect to the local feature size of the surface, the RDT provides a good approximation of the Hausdorff distance to the surface. Moreover, it provides a good approximation of normals, areas and curvatures. The meshing algorithm iteratively refines an initial 3D Delaunay triangulation until all surface Delaunay balls meet some properties. More specifically, starting from a small set of points on the surface, the method inserts the center of a *bad* surface Delaunay ball at each iteration (note that this center corresponds to the intersection between the Voronoi edge and the surface). A surface Delaunay ball is considered bad when it does not meet any of the following requirements:

- Angle bound (α_a): The RDT Delaunay triangle inside the ball must have all its angles larger than this angle bound.
- Radius bound (α_r): The ball must have a radius lower than this bound.
- Distance bound (α_d): The distance between the center of the ball and the circumcenter of the associated RDT triangle must be lower than this bound.

The algorithm terminates when the RDT contains no triangle with a bad surface Delaunay ball. The RDT is then the targeted approximation of the surface, and tuning the criteria defining bad Delaunay balls is a means of controlling the quality of the approximation and of the output isotropic surface triangle mesh in terms of sizing and shape of the triangles. The process of inserting these points is known as the *Delaunay refinement* procedure.

In our context, an interesting property of this meshing approach is that it only requires devising an oracle that, given a Voronoi edge, i.e., a line segment query, computes its intersection with the surface. This loose requirement makes the algorithm well-suited not just to isosurface meshing, but to various application scenarios. For instance, this technique can be applied to remesh an already existing model (see Figure 3.8). In this scenario, a triangle mesh is re-triangulated, in a way that its triangles follow the user defined restrictions imposed by the α_a, α_r and α_d parameters. This step may be useful in order to retrieve a more regular mesh, or an approximation of the surface at a different level of detail. Another example of the versatility of this method was proposed by Salman and Yvinec [180], where they used this surface meshing approach to recover a surface directly from an unstructured triangle soup. Note also that its coarse-to-fine procedure maintains a low memory consumption.

It should be noted that the proposals presented in Chapters 5 and 6 are directly related to this meshing methodology. However, there have been some variants and improvements in the literature. Cheng et al. [51] proposed a different approach for Delaunay refinement, also with theoretical guaranties, based on the topological ball property. The oracle for this method becomes a bit more complicated, since the approximated surface must be able to answer queries involving the previous line intersection test plus computations

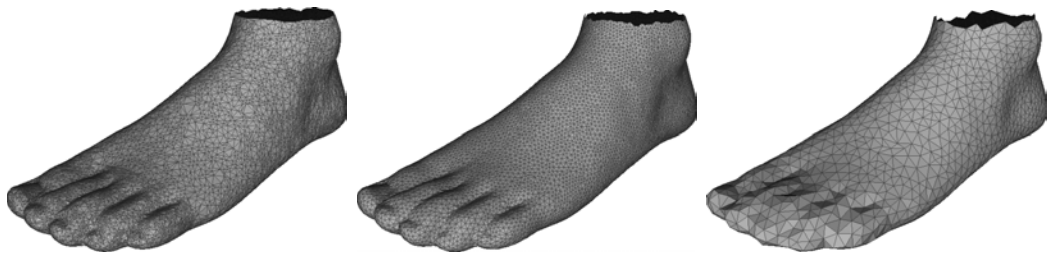


Figure 3.8: Sample of the behaviour of a surface mesher [30]. The original (irregular) mesh is on the left, and center/right show the results for two different parameterizations of the meshing algorithm, providing different resolutions for the input mesh. Note how the remeshed surfaces promote triangles closer to regular.

of *critical points* in a given direction and some specific *silhouette points*. On the other hand, there have been some proposals trying to diminish the memory overhead that the method requires for maintaining the Delaunay structure with very large models and/or high resolutions. This is the case of the method described in Dey et al. [67], where they propose a divide and conquer method in which different parts of the surface are reconstructed separately and then joined back together, maintaining coherence. Finally, a recent line of work is devoted to the recovery of piecewise-smooth surfaces. In order to deal with sharp edges, a common approach is to use the *protecting balls* mechanism proposed by Cheng et al. [50].

Finally, note that even if we focus on the surface meshing case, the Delaunay refinement algorithm is suitable for the meshing of domains with arbitrary dimensions. For instance, the method can be used in $\mathbb{R}^3$ to create a well-shaped tetrahedralization of a volume.

Chapter 4

State-of-the-Art Review

4.1 Introduction

There is a wide variety of 3D scanners in both nature and resolution. A 3D scanner is a device that measures 3D positions on an object's surface and returns an array of distance values. Local scans can then be joined into a single reference frame by means of a registration technique. Once all the scans are merged, the object is represented by discrete measures of its surface. This set of measures, which do not follow any underlying structure, is called a point cloud or point set. In our precise scenario, the scanner is a camera, and the point clouds are obtained through the techniques explained in Chapter 2.

Having the object represented as a point cloud gives us little information about the overall appearance of the object. The connectivity information between points is needed to get a notion of *visibility* for the different parts of the object given a specific point of view. Despite the fact that new emerging visualization techniques are trying to exploit the viability of the points as a surface primitive (e.g., see the work of [5]), the vast majority of these approaches are geared towards finding a continuous surface representation following the shape of the scanned object that the points describe. This process is referred to in the literature as the problem of *surface reconstruction from an unorganized set of points.* The point set is called unorganized because there is no assumption on the positioning of these points, i.e., they are not assumed to be following any regular structure and their connectivity is unknown. From now on, and to summarize, this problem is referred to in the text as the *surface reconstruction problem.*

It is worth remembering that the surface representation we are assuming as the result of these kinds of methods, as stated in previous chapters, is a triangle mesh. Also, we intentionally omit approaches working with piecewise-smooth surfaces. The problem of recovering surfaces with sharp features (edges and/or corners) is a research area that has gained popularity in recent years. However, our application scenarios are more geared towards the biology, geology and archeology areas, where sharp edges or corners are rare to find. For this reason, we base our work in the study and development of surface reconstruction methods working on smooth surfaces. It is obvious that in some cases we can find structures containing sharp edges in underwater scenarios, such as man-made structures (e.g., subsea manifolds), or specific species (e.g., some types of corals). Nevertheless, the overall surface of the objects can still be recovered using the assumption of smooth surfaces, even if these sharp edges may not be correctly represented.

Notice that the assumption of having no organization on the points gives generality to the problem, which broadens the possible scope of methodologies to apply and also their application fields. The areas of application include (among others): reverse engineering, virtual reality and automatic modelling. Furthermore, this problem has been studied by different research communities, including computer graphics, computational geometry and computer vision. In fact, one of the reasons why this problem is so open is because it is ill-posed: given a point cloud, one can make several proposals of surfaces that pass on (or near) the points. The problem is then bound to rely on the sampling of this surface, which translates into finding the surface that best fits the distribution of the set of points. It is then important to get a good sampling of the object, taking into account that, normally, high frequency areas of the surface (like edges or corners) need more samples to be reliably represented than smoother parts.

Normally, the input points are not assumed to have any other information associated to them other than their 3D coordinates. Nevertheless, some surface reconstruction methods use additional information on the points, like their normals, to help in disambiguating the problem. Furthermore, information does not necessarily have to come from the points, but can come from other sensors, such as methods using vision-based information (including textures or camera positions), to further restrict the search for a surface that corresponds to that of the real scanned object.

The motivation of the present chapter is thus to unify the main contributions from different communities devoted to solving the surface reconstruction problem of smooth surfaces. We present an extensive, self-contained review of the methods and divide them according to common properties. We start by briefly overviewing the challenges present in real world point cloud datasets. We then describe the proposed classification in Section 4.3, while the main groups are further discussed in Sections 4.4 and 4.5. Finally, Section 4.6 presents some conclusions regarding present approaches and future trends expected in this field.

4.2 Challenges of Point Set Data

Despite the advances in accuracy and methodology of range scanning sensors, they are bound to suffer measurement errors. Depending on the type and nature of the scanner, these errors are more or less pronounced, but they are always present since the scanning process is not perfect. It is important for surface reconstruction methods to be aware of this fact. We will see in the sections to come that some methods assume the data as being noise-free, which bounds the user to use a preprocessing step on the input points before applying the surface reconstruction method. On the contrary, some other methods assume the data to be noisy, and are able to recover a surface despite the errors in the input.

There are two main types of errors when talking about point clouds: noise and outliers. *Noise* errors are related to the quality of the measurement, that is, they are variations caused by the precision and repeatability of the sensor used to get the samples of the surface. If a good knowledge of the sensor is provided, these errors can be modeled by a Probability Density Function (PDF). However, it is possible to have different levels of noise within a given dataset. On the other hand, *outliers* are wrong measurements, meaning that they do not represent a sample of the surface, which normally makes them lie far from their neighbors. These kinds of errors are caused mainly by defects in the scanning procedure. For instance, a laser scanner applied to a reflective surface may not be able to get the samples on the surface, and will retrieve these *false* measurements as correct data (although these measurements correspond to outliers). In the specific case of optical-based methods, the level of noise and outliers is normally higher than in the case of other methods, like laser-based range scan. Specially when applying these methods underwater, the poor imaging conditions complicate the modelling process, which results in a wider variability of the noise and a larger number of outliers than when the point set reconstruction is applied in a more controlled scenario. It is thus important to take into account these aberrations when dealing with surface reconstruction on this type of data.

Besides noise and outliers, other challenges present in the point set come from the scanning methodology used. For instance, some parts of the object may be under-represented or even not scanned directly. In these cases, the resulting mesh depends on the surface reconstruction method used. On the one hand, some methods consider holes as missing data, and try to cover these holes in the surface by filling them in a plausible way. On the other hand, some methods assume that the surface may be bounded, and use the lack of data as a means to detect such boundaries. Moreover, another important source of error is the registration of the different scans in a global frame. Obviously, a given 3D object needs to be scanned from different points of view in order to be reconstructed. However, if the registration of these individual scans in a global frame is erroneous, the point set

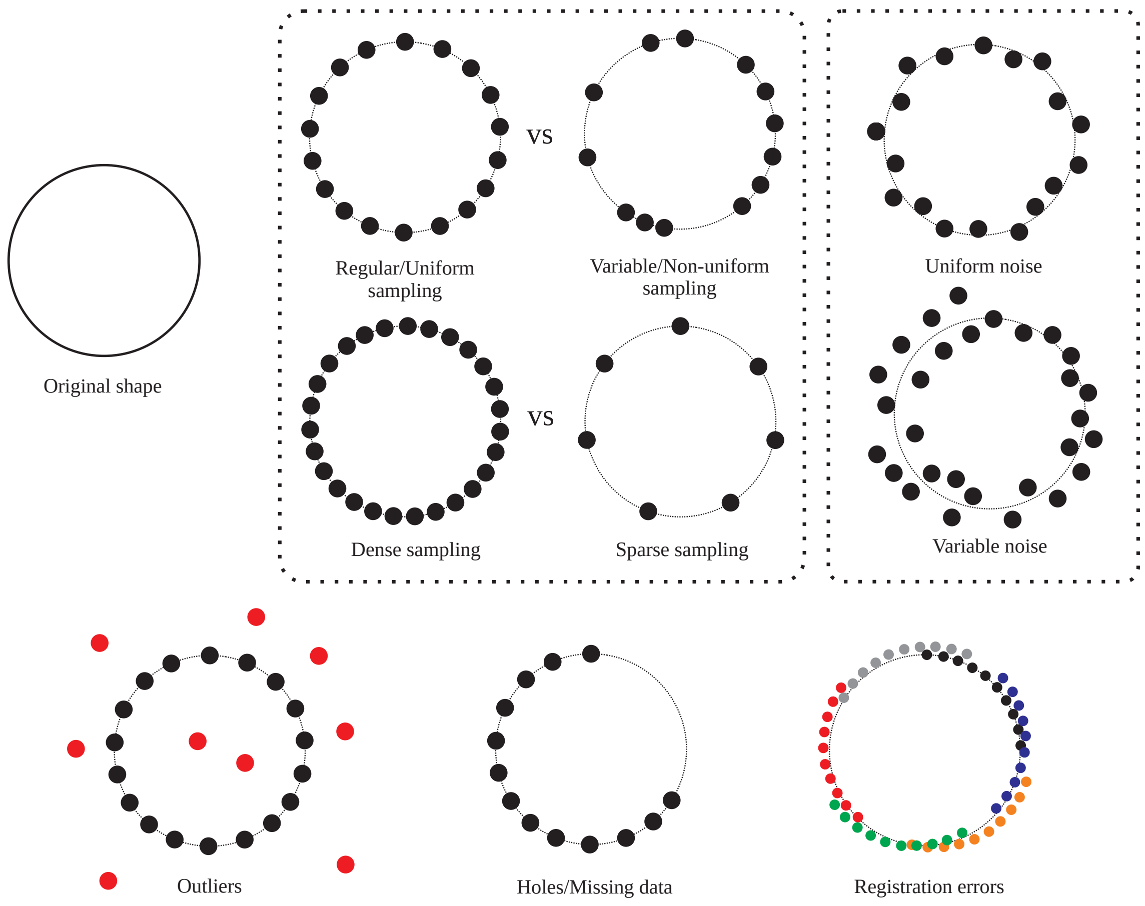


Figure 4.1: Common challenges in real point based datasets to be handled by surface reconstruction approaches.

will present some errors, such as *double contours*.

Finally, for a graphical summary of the problems associated to real-world datasets, the reader is referred to Figure 4.1.

4.3 Classification

The problem of surface reconstruction is a continual hot research topic nowadays in various research areas. The vast number of approaches proposed by these different communities, following different methodologies and heuristics, complicates the creation of a global classification able to include them all.

There have been some proposals in the past to propose a classification on surface reconstruction methods. However, surveys dealing with the problem in general are outdated, and some of them are partial, focusing only on some subset of the methods. One can find early discussions about the surface reconstruction problem within surveys dealing with range data interpolation, but these reviews normally treat the problem of surface interpolation for the values on a single range image, like in Bolle and Vemuri [32], or they focus on the interpolation-based procedure between vertices when having an already triangulated domain, as in Lodha and Franke [144].

On the one hand, the work of Mencl and Müller [157] has been considered by many authors as the first survey dealing with the surface reconstruction problem strictly using an unorganized point cloud as input. They divide the methods according to common basic strategies: *spatial subdivision*, *distance functions*, *warping*, and *incremental surface growing*. The problem of this classification is that it is not exclusive, i.e., a method can fall into more than one category, since more than one of these basic strategies may be used in an algorithm. On the other hand, Cazals and Giesen [44] propose another classification focused on methods using the Delaunay triangulation as their base: *tangent planes*, *restricted Delaunay*, *inside/outside labeling* and *empty balls* methods. Despite being a clear classification (we use some similar categories in our proposal), the great quantity of literature on methods not relying on Delaunay are omitted. Finally, a short survey focused on recent developments is presented in Schall and Samozino [185], where methods are divided into *Delaunay-based*, *implicit surface interpolation* and *learning-based methods*. However, they miss the previous literature leading to these methods and, given its publication date, nowadays it is somewhat outdated.

It is obvious that an up-to-date generic taxonomy of surface reconstruction methods is missing in the literature. We propose one inspired by what other authors proposed in the past, by using some of their ideas to build our own classification, and also a categorization able to encompass all the methods in the state of the art. Even if not unique, in the next section we propose a classification including up-to-date proposed approaches. Detailed descriptions of the methods falling in each category are given throughout the present chapter.

4.3.1 Proposed Classification

We aim at finding a global classification providing an overview of the different methodologies proposed to solve the surface reconstruction problem. The basis of our classification is to separate the methods according to their approaches; *interpolation based* or *approximation based.*

We understand by interpolation-based approaches those methods whose output surface contains all or part of the input points as its vertices. On the other hand, we refer as approximation-based methods to those using the input points as a notion of where the surface is, and whose output surface does not necessarily have the input points as part of its vertices. Consequently, in the text we distinguish between interpolating surfaces $\bar{S}$, and approximating surfaces $\hat{S}$. The reason for this main classification is due to the common preprocessing and post-processing associated with these two approaches.

The surface resulting from an interpolation-based method adapts to the density of the input point set, since they are part of the vertices of the surface. However, the quality of the final surface is bounded by the quality (i.e., the noise) of the input point cloud. For this reason, these kinds of methods should be applied when the input point cloud is considered ideal (or close to it). Normally, a smoothing step is required to improve the quality of the final surface. This smoothing can be applied in preprocessing in the input point cloud, or in post-processing in the resulting mesh. In the case of using a surface reconstruction method that does not take into account noisy inputs, a preprocessing in the input points is needed. On the other hand, if a method allowing noisy input is applied, a post-processing mesh smoothing step may be required to improve the visual appearance of the mesh.

Additionally, interpolation-based methods are normally applied to input points having further information associated. The type of information related to each point depends on its origin, i.e., the scanning methodology used to gather them. For instance, in multiple-view stereo computer vision 3D reconstruction systems, each point contains information such as the views that have generated it, and its 2D coordinates in each image, texture description, etc. If this information is needed in further processing after the reconstruction step, it is important to keep them as being part of the final surface. In the previous example, the information provided could be directly used easily to get a textured model out of the images of the multiple-view stereo system without further processing.

On the other hand, approximation-based methods are applied when a *smoother* result is needed. Since points are normally used only as a notion of where the surface should be, they normally mitigate the effect of noise in the data. Nevertheless, their resilience to noise does not ban the use of preprocessing on the points to obtain better results. Notice that interpolation-based methods are bound to return an explicit representation of the surface, since triangles must pass through the input points. This is not the case in approximation-based methods, where a majority of approaches provide an implicit formulation of the surface as result. As previously discussed, a surface mesher approach like those presented in Section 3.4, is used in these cases to get the final surface in explicit form.

Of course, these two groups contain a great number of methods, and a finer division should be applied. Further sections present how these approaches can be refined according to common methodology.

Tables 4.3 and 4.4 present a summary of the proposed classification for the methods available in the state of the art (ordered by their appearance in the text). In addition to the main classification, the following important properties of the methods are discussed in the left-hand columns of the tables:

- **Boundaries:** Indicates whether the method is able to detect and reconstruct boundaries on the surface. Note that, in the opposite case, the method fills the possible holes.
- **Noise:** Denotes if the method is able to deal with noise in the input data. A discussion on the type and level of the noise assumed is given in their corresponding sections in the text.
- **Outliers:** Ability of the method to deal with a reasonable amount of outliers.
- **Guaranties:** Theoretical guaranties of the method. Some methods, mostly in the Computational Geometry field, are able to provide theoretical guaranties for the correctness of the method. These guaranties are normally based on restrictions regarding the sampling of the surface. Notes on the assumed sampling and the guaranties of each method are commented on in the text.
- **Additional Info.:** Indicates whether the method uses additional information during the reconstruction process other than the coordinates of the input points. This additional information is normally associated to the input points (e.g, normals), but other information related to the sensor used to gather the data may also be used. If the method requires per-point normals, but a heuristic to compute them is given in the original article, we consider that no additional information is used.
- **Complexity:** Overview of the complexity of the method, as discussed by the author. Taking into account the different implementation possibilities of each method, we decided to include this information only when the authors discus it in the original paper.

In order to build a coherent review, we use common nomenclature across all the methods. Furthermore, when possible, we provide results of applying the presented algorithms to publicly available datasets. Table 4.1 defines the origin of the tested methods, which can be either the original implementation (when provided by the authors), a third party implementation or our own implementation. Regarding the input point sets used in the tests, when raw point sets are used, they are plotted in a single color, but when per-point normals are used, this information is used for shading. Note that while these tests may be useful for an easy interpretation of the behaviour of each algorithm, they do not represent an analysis of the advantages/weakness of each method, which are otherwise noted in both the text and Tables 4.3 and 4.4 (at the end of this chapter).

Table 4.1: Sources of the implementations of the methods used in this chapter.

Method	**Implementation**	**Available at**
α-shapes [72]	CGAL [1]	`http://www.cgal.org/Manual/latest/doc_html/cgal_manual/Alpha_shapes_3/Chapter_main.html`
Crust [10](first part)	MeshLab [53]	`http://meshlab.sourceforge.net/`
Ball-Pivoting [23]	MeshLab [53]	`http://meshlab.sourceforge.net/`
Smooth Greedy [54]	Ours	-
Zipper [154]	MeshLab [53]	`http://meshlab.sourceforge.net/`
Power Crust [13]	Original	`http://www.cs.ucdavis.edu/~amenta/powercrust.html`
Robust Cocone [65]	Original	`http://www.cse.ohio-state.edu/~tamaldey/cocone.html`
Tight Cocone [64]	Original	`http://www.cse.ohio-state.edu/~tamaldey/cocone.html`
Peel [66]	Original	`http://www.cse.ohio-state.edu/~tamaldey/Peel.html`
Graph Cuts Stereo [132]	Ours	-
Hoppe Method [104]	Ours	-
Markov Random Field [169]	Original	`http://www2.imm.dtu.dk/image/MRFSurface/download.html`
MPU [165]	Original	`http://www.den.rcast.u-tokyo.ac.jp/~yu-ohtake/software/index.html`
Adaptive MPU+RBF [167]	Original	`http://www.den.rcast.u-tokyo.ac.jp/~yu-ohtake/software/index.html`
Multi-scale MPU+RBF [166]	Original	`http://www.den.rcast.u-tokyo.ac.jp/~yu-ohtake/software/index.html`
Smooth PU [161]	Original	`www.den.rcast.u-tokyo.ac.jp/~nagai/Material/PoissonPU/PoissonPU.zip`
Point Set Surfaces [5, 2]	As part of APSS package	`graphics.ethz.ch/apss/`
Implicit PSS [124]	As part of APSS package	`graphics.ethz.ch/apss/`
Algebraic PSS [97]	Original	`graphics.ethz.ch/apss/`
Touch-Expand Graph Cuts [135]	Original	`http://vision.csd.uwo.ca/code/`
Fourier Transform [117]	Original	`http://www.cs.jhu.edu/~misha/Code/Reconstruct3D/`
Wavelets [153]	Original	`http://josiahmanson.com/research/wavelet_reconstruct/`
Poisson [118, 119]	Original	`http://www.cs.jhu.edu/~misha/Code/PoissonRecon/`

Table 4.1: Sources of the implementations of the methods used in this chapter (continued).

Method	Implementation	Available at
Smooth Signed Distance [41]	Original	http://mesh.brown.edu/ssd/software.html
VRIP [60]	Original	http://grail.cs.washington.edu/software-data/vrip/
Spherical cover [168]	Original	http://www.den.rcast.u-tokyo.ac.jp/~yu-ohtake/software/index.html

Table 4.2: Sources of the point sets used through this chapter.

Repository	Dataset name	Figures
Stanford Scanning Repository[1]	Armadillo	4.9
	Dragon	4.16
	Happy Buddha	4.17
	Bunny	4.18
H. Hoppe[2]	mechpart.4102	4.2
	cat10.10000	4.34.11
	mannequin.12772	4.5
Aim@Shape[3]	Foot	4.44.6
	Bimba Con Nastrino	4.7
	Olivier's hand (Kreon)	4.8
	Max Planck	4.124.20
	Gargoyle	4.15
	Bust - watertight	4.13
Y. Furukawa (Multi-view Stereo)[4]	SkullA	4.104.19
Y. Nagai[5]	Hotei	4.14

[1] http://shapes.aimatshape.net
[2] http://research.microsoft.com/en-us/um/people/hoppe/thesis_data.zip
[3] http://graphics.stanford.edu/data/3Dscanrep
[4] http://homes.cs.washington.edu/~furukawa/research/mview/index.html
[5] http://www.den.rcast.u-tokyo.ac.jp/~nagai/material.html

4.4 Interpolation-based Methods

Interpolation-based methods are divided into two main groups corresponding to those using a *surface oriented* or a *volume oriented* approach.

Surface oriented approaches solve the surface reconstruction process by retrieving the surface directly, that is, joining the input points in triplets in order to form the triangles belonging to the surface, and then merge these triangles into a single coherent mesh (usually manifold). On the other hand, volume oriented approaches cast the reconstruction process as obtaining the boundary between two volumes of space: the *inside* and the *outside* of the scanned object/scene. In other words, surface oriented methods work directly with the construction of triangles, while volume oriented ones deal with the separation of the space into two volumes, and the surface is defined as the triangles being at the interface between them.

A great majority of the interpolation-based methods come from the Computational Geometry field and use the Delaunay triangulation and the Voronoi diagram structures as their base. Moreover, by their nature, most of the volume oriented methods provide a volumetric representation of the object of interest, and not just a description of its surface. Hence, these last methods assume watertightness on the surface to be reconstructed, and consequently prevent the recovery of bounded surfaces (with some exceptions).

A classification of the main contributions to this group of methods is shown in Table 4.3. In the following sections, a more comprehensive review of the methods forming each category is presented.

Several approaches have been proposed for each surface/volume oriented method group, so we propose a second subdivision of these classes focusing on common methodologies.

4.4.1 Surface Oriented

Surface oriented methods are divided into the following groups:

- **Delaunay Triangle Selection:** Starting from the 3D Delaunay triangulation of the input point cloud, a subset of triangles following some properties are selected as potentially being part of the surface. Then, a greedy post-processing step is applied to get the connectivity between these triangles describing a manifold surface.

- **Surface Growing:** Incrementally add new triangles to a surface that is growing at each iteration, one triangle at a time. The selection/creation of the new triangle to insert next is guided/restricted by the currently growing surface.

- **Integration:** Some connectivity relationships between points is extracted from the information provided by the scanning sensor, and these local reconstructions are then merged together into a single surface model.

4.4.1.1 Delaunay Triangle Selection

Delaunay Triangle Selection methods differ mainly in the two heuristics that these methods follow to extract the surface: the heuristic that defines a triangle as being a candidate to be part of the surface, and the heuristic defining how to join these candidates into a global consistent surface. Notice that these two steps, as opposed to Surface Growing methods, are not really dependant on one another.

α-shape The first approach in this category was devised by Edelsbrunner and Mucke [72], where they introduce α-shapes. The original definition of an α-shape, given in Section 3.3.6.2, only defines a simplicial complex, containing tetrahedra, triangles and edges. Obviously, this does not cope with the desirable surface representation as a manifold triangle mesh. In the proposal by Guo et al. [98], they propose a heuristic to extract the outer surface bounding the simplicial complex. Given the α-shape, a normal for each face in the complex is computed. Since only undirected normals can be extracted from triangles, a coherent orientation between them is obtained through the method of Hoppe et al. [104] (further comments in Section 4.5.1). Once oriented normals are available, a greedy procedure is used to extract the outer surface. Starting from an arbitrary triangle, faces are added to the surface if they share an edge with a triangle already in the surface and the dihedral angle between them is $dihedral(t_i, t_j) = \pi + \epsilon$, where ϵ is a parameter (in the paper suggested to be $\epsilon = \pi/3$). It is obvious that this approach has the problem of requiring user input on the α parameter. It is worth noticing that an optimal α parameter may not exist for a given point cloud with variable sampling, since α-shapes assume regular sampling.

Normalized mesh The α-shapes are not the only proposal for a simplicial complex resembling the shape of an object from point clouds. In an early work of Attali [15], another simplicial complex called the normalized mesh is presented. It introduces the concept of r-shapes, defined as those morphologically open and closed with respect to a disk of radius $r > 0$. While they provide a sampling condition and prove its correctness in a 2D case, a 3D case follows some heuristics. Basically, they retain a triangle as part of the surface if the intersection between the enclosing balls defined by their 2 associated triangles is larger than a threshold (quite similar to the 2D β-skeleton definition [122]). This triangle selection recovers most of the shape under ideal conditions, but some holes and inconsistencies remain. A heuristic is then proposed to complete the surface. All the triangles in Del(P) are put in a queue and, at each iteration, a pair of tetrahedra is merged into a single triangle providing that none of the previously selected triangles disappear and the points remain being in the boundary of the object (i.e., it is not allowed

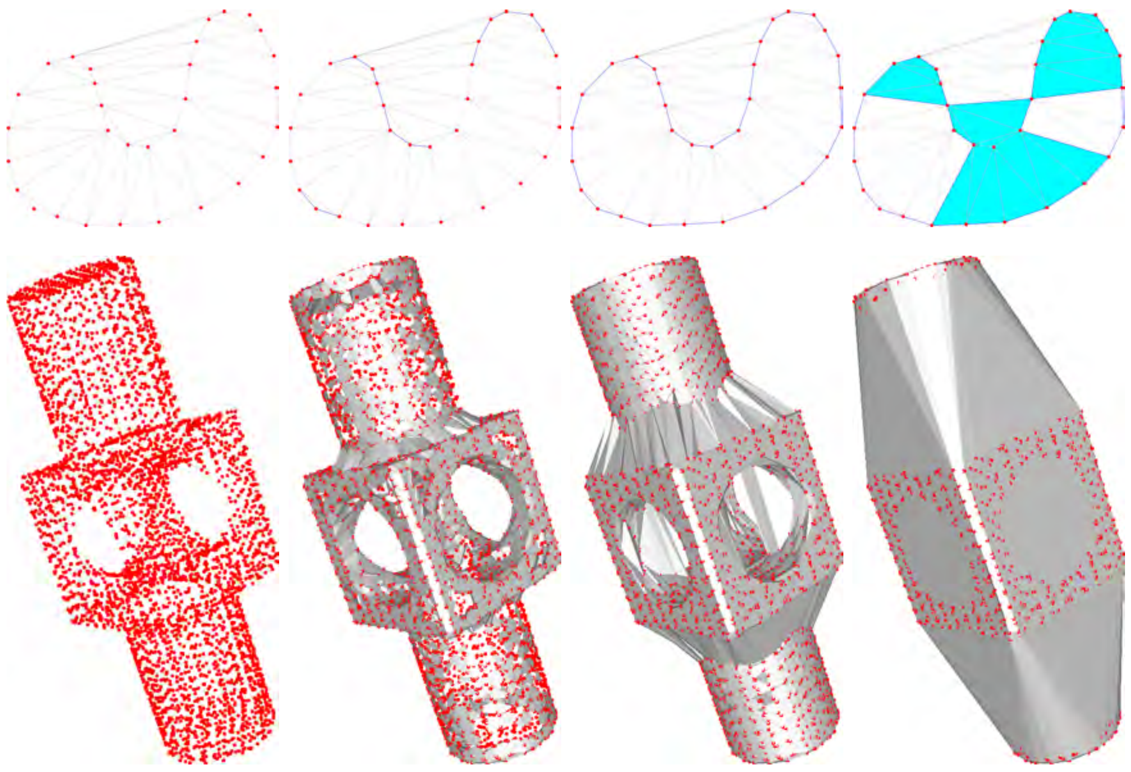


Figure 4.2: The α-shapes [72] results for different values of α, increasing from left to right (first row for 2D and second for 3D). In the 2D case, the Delaunay triangulation is marked in light gray, in order to show that the alpha shape is a subset of it. Note how alpha equals the point set itself for a sufficiently small α, and converges to the convex hull of the points for a sufficiently large value.

to have isolated points).

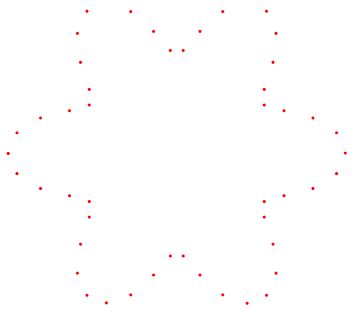
Crust Very close in time, Amenta and Bern [10] proposed a method which is known for being the first to provide theoretical guaranties in 3D: the Crust algorithm.

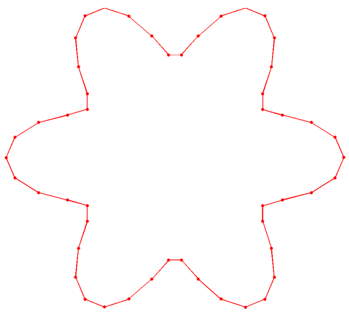
The authors first proposed a 2D curve reconstruction version in the paper [11], where they defined the curve to be the subset of edges in $\mathrm{Del}(P)$ intersected by their dual Voronoi edges. As stated in Section 3.3.7, the medial axis is directly related to the shape of the object. Also, they observe that the Voronoi diagram can be seen as an approximation of the medial axis: vertices in the Voronoi diagram define a set of maximally empty spheres, which resemble the empty balls in the definition of $\mathrm{MA}(P)$. Then, the edges of the curve are defined as those having a circumsphere empty of other vertices in P (already achieved if in $\mathrm{Del}(P)$) and empty of vertices of $\mathrm{Vor}(P)$.

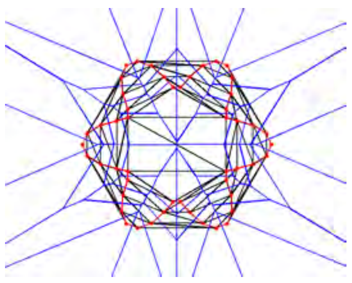
In the 3D case, points in $\mathrm{Vor}(P)$ may fall too close to the surface, preventing the method to be directly portable to $\mathbb{R}^3$. However, they observe that just a subset of the vertices in $\mathrm{Vor}(P)$ do resemble the medial axis, which are the poles (see Section 3.3.6.1). Thus, the method now consists of selecting those triangles whose enclosing ball does not contain any other point from P nor a pole from $\mathrm{Vor}(P)$. Thus, the algorithm basically computes $\mathrm{Vor}(P)$, selects the poles $Q(P)$, and then computes a final $\mathrm{Del}(P \cup Q(P))$. From this last triangulation, the triangles having all three vertices from P follow the previously stated property.

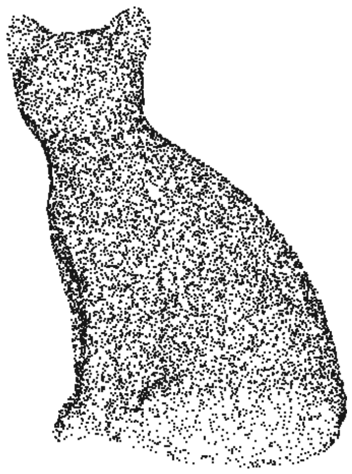
This method gives theoretical guaranties provided the points follow the ϵ-sampling for a sufficiently small ϵ. However, this sampling condition is not likely to happen in real data. For this reason, a final set of heuristics is needed. First, a filtering step is applied, consisting of deleting triangles whose normals produce a large angle with the line joining each of its three vertices p_i, p_j, p_k and their corresponding positive poles q_i^+, q_j^+, q_k^+ (the notion of *large* being governed by a user's parameter). Then, a final step responsible for orienting the normals coherently and constructing a manifold mesh is applied. By taking a triangle on the convex hull as the reference triangle, its q_i^+ defining the direction of the normal, its orientation is propagated by iteratively visiting new incident triangles in a breath first manner. Figure 4.3 shows a sample of the behaviour of the Crust when applied to both 2D and 3D datasets.

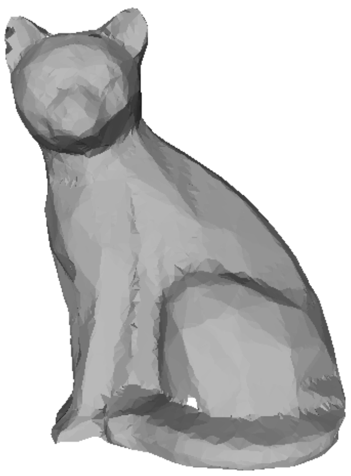
Cocone Improving on the Crust proposal, Amenta et al. presented the Cocone algorithm [12], also relying on the ϵ-sampling to get theoretical guaranties. For an ϵ-sampling with $\epsilon \leq 0.06$, the algorithm is proven to provide a manifold mesh interpolation $\bar{S}$ homeomorphic to S, and any point on $\bar{S}$ is also bound to be at a distance of at most $\frac{1.15\epsilon}{1-\epsilon} lfs(p_i^S)$ for some point $p_i^S \in S$.


(a) 2D Point Set


(b) 2D Crust


(c) 2D Crust + Del(P) + Vor(P)


(d) 3D Point Set


(e) 3D Crust

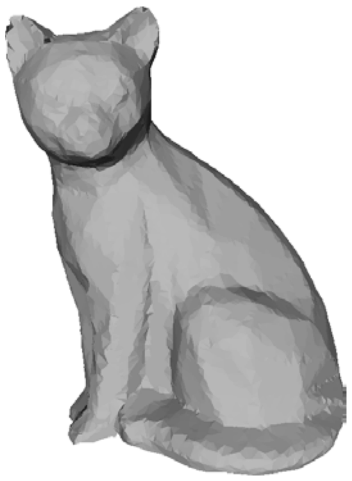

(f) 3D Cocone

Figure 4.3: Examples of the Crust [10] and Cocone [12] algorithms. The first row shows the behaviour of Crust in 2D (from left to right: point set, crust and crust with the corresponding Delaunay triangulation and Voronoi diagram superimposed). Note how the curve is made up of those segments intersected by their Voronoi dual. The second row shows the reconstruction of the Cat point cloud (left to right: point set, Crust, and Cocone). In both cases, and despite most of the cat's shape being correctly recovered, some holes are present in the surface because of the sampling conditions not being fulfilled.

In this algorithm, the line joining p_i with its positive pole q_i^+ is used as an estimation of the normal at that point. Then, the Cocone is defined to be the cone-complement with its apex at p_i that makes an angle of $\pi/2 - \theta$ with the normal estimation at p_i. Edges in Vor(P) intersected by the three cones of the three Voronoi regions that formed that edge are selected. A set of candidate triangles are generated as the set of dual triangles corresponding to the selected Voronoi edges. Starting from those candidate triangles, a manifold extraction step is required. This step consists of recursively removing from the candidate triangles those adjacent to *sharp edges*. A sharp edge is defined as that whose adjacent triangles form an angle greater than $3\pi/2$. The deletion of sharp edges would recursively delete boundary triangles, since boundary edges are sharp edges, which would result in deleting the entire surface. Even though this algorithm is supposed to reconstruct a watertight surface, undersampling normally also produces holes in the reconstructed set of triangles, so this has to be taken into account. For this purpose, they perform an umbrella check for each point p_i, which consists of checking if the triangles adjacent to it form a topological disk, and no two consecutive triangles around the disk make a dihedral angle of less than $\pi/2$ or more than $3\pi/2$. A triangle incident to a sharp edge is deleted only if its three vertices are part of an umbrella. Notice that the Cocone method requires a single Delaunay triangulation, as opposed to the Crust algorithm, which requires two. This makes this method less expensive computationally. A visual comparison between Crust and Cocone is presented in Figure 4.3.

The Cocone algorithm has many variants that deal with some important properties for reconstruction. It was first extended in Dey and Giesen [62] to deal with boundaries, by defining a boundary as an undersampled region. It was also modified in Dey et al. [63] to deal with datasets consisting of large amounts of points, by resorting to a partition and merging strategy: Space is partitioned following an octree structure, and the Cocone algorithm is applied locally at each of its leaves. To get a consistent merging of all the local Cocone surfaces, a band of points in adjacent octree bounding boxes are taken into account when building the local reconstruction at each leaf. Volumetric variants of this method are presented in Section 4.5.

Gabriel The approach of Petitjean and Boyer [171] proposes yet another property for selecting triangles belonging to the surface, by claiming that *good* candidates tend to follow the Gabriel property. We will see in this chapter how many methods in the state of the art take advantage of the Gabriel property. For noise-free datasets, Gabriel triangles inside the Delaunay triangulations are likely to be part of the surface of the object, as can be seen in Figure 4.4.

Again, this method relies on a sampling condition for their method to work, but in

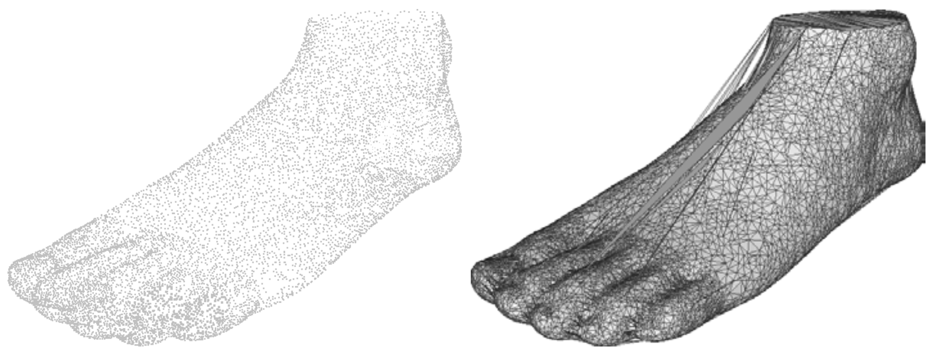


Figure 4.4: Gabriel triangles (right) of a given point set (left). Note how the majority are part of the surface of the object.

this case the condition is only related to the input point set, rather than being defined with respect to the unknown surface S. Notice that the sampling conditions presented in the previous algorithms (ϵ-path and ϵ-sampling) are defined over a surface that we do not know, and thus we cannot conclude if a given point set P fulfills the sampling criteria or not. On the contrary, a sampling condition built on the samples alone can be checked before the reconstruction.

The criterion is based on two main concepts: Local Granularity and Local Thickness. The Local Granularity corresponds to the radius of the largest of the balls circumscribed to the simplices in the star of a point p_i. On the other hand, the Local Thickness is the distance from a point p_i to the discrete medial axis. The discrete medial axis, as intuitively introduced in the Crust algorithm, is formed by the closure of centers of empty Voronoi balls. Once these concepts are defined, a surface is said to be a regular interpolant of P if the local granularity is strictly smaller than the local thickness for all its vertices. Consequently, the point set is regular if it admits at least a regular interpolant. The algorithm computes the surface as a subset of the 3D Gabriel Graph $GC(P)$, constructed by incrementally selecting triangles while minimizing granularity at their vertices.

Umbrella Filter Another example based on the 3D Gabriel graph is the Umbrella Filter proposed by Adamy et al. [3]. They define the λ-complex as a subcomplex of a Delaunay triangulation defined as a combination of the α-shapes and the β-skeleton. Given a simplex u, a λ-ball is an open d-dimensional enclosing ball having a diameter $diam(u)$ that has all the vertices on its boundary. The λ-interval defines the range of λ-balls (at least one possible) empty of points that each triangle inside Del(P) can generate. Thus, in this case, the λ parameter defines a *range* of values, as opposed to the single-valued α and β in the previously mentioned complexes.

Given a triangle t, its two associated tetrahedrons tet_1 and tet_2 inside Del(P) define its possible λ range as $\lambda_i = \frac{diam(B(tet_1))}{diam(B(tet_2))})$. The lower bound of the λ-interval is $\min(\lambda_1, \lambda_2)$, while the upper bound is 1 if the centers of the circumscribed balls $B(tet_1)$ and $B(tet_2)$ lie on different sides of the hyperplane defined by t.

While this definition is useful for proposing a correct solution in the 2D case, in 3D they take advantage of the Gabriel graph. Notice that, by definition, the Gabriel Graph is inside the λ-complex with interval $\lambda = [1, 1]$. The Umbrella Filter algorithm chooses an optimal umbrella from the Gabriel graph for every sample point, where an umbrella corresponds to a union of a set of triangles incident to a point p_i on the surface mesh that is homeomorphic to a disk. The optimal umbrellas are those minimizing the maximum of the lower λ-interval bounds associated to its triangles. The resulting surface may present non-manifold configurations that are heuristically filtered in a post-processing step.

Flow Complex The last representative example in this section also relies on Gabriel triangles to get an initial description of the shape. The Flow algorithm presented by Giesen and John [88] builds on the idea of associating a dynamical system induced by the sample points to the 3D Gabriel complex. This dynamical system is built on the distance function, $dist_{flow}$, that assigns to each point its least distance to any of the sample points. The Flow, or the dynamical system that rules the function $dist_{flow}$, is used to get the reconstruction. The critical points of this function can be determined from the Voronoi diagram and the Delaunay triangulation of the set of points. Each of these critical points has a label according to the Dimension of the Delaunay object used to generate it:

- **Index** 0**:** Minima, which correspond to the input points.
- **Index** 1**:** Intersection of a Delaunay edge and its dual Voronoi facet.
- **Index** 2**:** Intersection of a Delaunay facet and its dual Voronoi edge.
- **Index** 3**:** Local maxima of the distance function.

From this function, the interesting parts are the stable manifolds of index 2 saddles. That is, the set of all points that *flow* (are attracted) into an index 2 critical point. The stable flow complex is given by the simplicial complex built from the 3D Gabriel graph and the triangulates whose points flow into index 2 saddles. From this set of triangulated cells, a manifold surface is extracted. The processing is based on pairing and cancelation of critical points. The pairing is between index-2 saddles and critical points of index 3 (maximums). A pair is valid if the pair formed by a saddle 2 a and a maximum of index 3 b in the stable manifold of a is contained in the boundary of the stable manifold of b. A valid pair (a, b) is eliminated if there is a Gabriel edge in the boundary of the stable manifold which is incident to the stable manifolds of at least three maxima. The pairs are canceled iteratively until no more pairs fulfill this condition. The resulting complex is called the *reduced flow complex*. A final step that ensures topological correctness is needed to eliminate parts of the surface that meet at a point or at an edge. A volumetric algorithm exploiting the same flow relations presented in this algorithm is described in Section 4.4.2.2, and a more detailed description of the flow complex is presented in a second paper by Giesen and John [89].

4.4.1.2 Surface Growing

Surface Growing algorithms recover the surface by building it incrementally, making local decisions at each iteration. As opposed to the methods in Section 4.4.1.1, Surface Growing methods do not have a first step recovering candidate triangles. Candidate triangles can be defined according to some properties, but their selection/creation is always dependant on the triangles that are at a specific moment on the surface being greedily constructed. Surface Growing methods can be seen as following a procedure that merges the two steps in the Delaunay Triangle Selection methods. Furthermore, most of them do not need to rely on the Delaunay triangulation as an underlying structure from which to extract the triangles: they are constructed *on the fly*.

Boissonnat's Greedy The first approach in this category is contained in an early article by Boissonnat [28], which is considered the first article in the literature dealing with the problem tackled in this thesis. From the two methods proposed in the paper, we focus here on the first. It is based on differential geometry properties and, more precisely, on the fact that the local neighborhood of a point in a differentiable surface can be approximated by its tangent plane. This means that we can create and decide the triangle to insert next on the surface by projecting the situation onto a local tangent plane, and making the decision there.

First, the k-nearest neighbors $K(p_x)$ of each point $p_x \in P$ are selected. The propagation in this algorithm is edge based, i.e., the principal primitive is the edge, so the contour of the current surface is tracked at each step. Starting from a first edge $e_{i,j}$ (not specified), a projection plane is constructed using $K(p_i)$ and $K(p_j)$ using a least-squares method. Then, the edge and neighboring points are projected onto the local tangent plane and a new point is selected to create a new triangle. The point to insert is the one *seeing* the edge on the surface with a maximum angle, that is, the one that creates a more regular triangle. Then the new triangle is added to the surface, the set of free points and the list of contour edges is updated, and the process is repeated on an unvisited edge (the order followed to pick up the next edge is not specified).

Note, however, that if the surface described by P contains areas of high curvature, the neighbors of a given edge may not be projectable on a plane. Besides, the nearest neighbor search does not assure neighbors found being real neighbors of the original surface S. Furthermore, the unrestricted growing of the surface could lead to triangles crossing one another. Despite its flaws, this method presents the basis of many methods in the present group.

Graph Greedy Another idea, presented by Mencl and Müller [156], is to use a graph as the underlying structure where the surface is built incrementally. The main idea is to augment the Euclidean Minimum Spanning Tree (MST) of the input points ($MST(P)$) incrementally. Even though this method proposes a well described processing pipeline, its execution depends on a large number of parameters. Their tweaking and the large number of steps required increases the complexity of this algorithm.

Ball-Pivoting Another important contribution is the Ball-Pivoting algorithm, presented by Bernardini et al. [23], especially for its simplicity and its low computational complexity. These two properties make this method applicable to large input sets, prohibitive at the time the algorithm was created. In the Ball-Pivoting, a triangle is part of the surface if a ball of a user-specified radius ρ touches its three points without containing any other point.

Starting with a seed triangle, the ball pivots around an edge until it touches another point, forming a new triangle. The output mesh is a manifold subset of an α-shape, as clearly the parameter ρ has a similar meaning to that of α. Following the previous observation, the manifold obtained is also part of the $\mathrm{Del}(P)$. However, and in contrast to the α-shapes algorithm, it does not need to compute explicitly the Delaunay triangulation, thus avoiding the complexity and memory requirements of building this structure.

The original algorithm needs an estimate of the normal at each surface sample to disambiguate cases produced by error in the measurement or registering process. The process begins by selecting a seed triangle. This initial selection is performed by getting a point p_i along all unvisited points. Then, pairs of points in the neighborhood of p_i, taken in distance order, are used to build candidate triangles. Once a candidate triangle made of consistent normals at each vertex has an empty ball of radius ρ, it is selected as the seed triangle. During the pivoting step, the ball remains in contact with an edge $e_{i,j}$ in the boundary. Let the center of this ball be $center(B(e_{i,j}))$. This restriction makes the circular trajectory of the ρ-ball constrained to a circle having as its center the midpoint $m_{e_{i,j}}$ of the edge $e_{i,j}$ and radius $\|center(B(e_{i,j})) - midpoint(e_{i,j})\|$. Then, points in the 2ρ neighborhood of $midpoint(e_{i,j})$ are considered. ρ-balls are placed on neighboring points, and the intersection points between the balls and the trajectory circle are the candidate

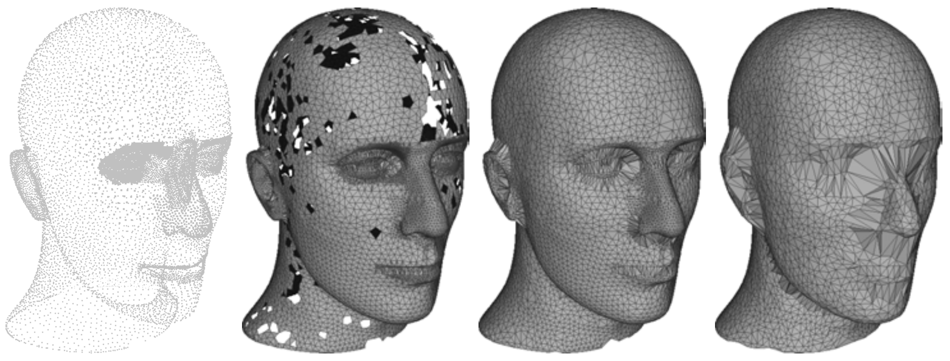


Figure 4.5: Behaviour of the Ball-Pivoting algorithm [23] for a changing value in the ρ parameter. The left-hand figure shows the input point cloud, and the value of ρ is increased starting from the second figure, from left to right. Note that, as was the case for the α-shapes, an optimal ρ value may not exist for a given dataset.

centers for the ρ-balls. The first center found following the circular trajectory is used to build the next triangle. New triangles are generated using a *Join* operation, adding a new triangle to the mesh, or a *Glue* operation, removing coincident edges.

It is worth noting how the speed and low memory footprint of this method lead to the application of the surface reconstruction to large datasets, containing millions of points. Furthermore, the proposal of Digne et al. [69] demonstrates that interpolation-based methods can be applied to noisy inputs. In their approach, a scale space version of the point set is built, and then the ball pivoting algorithm is applied in a smooth scale. Then, through back-tracking in the scale-space, the vertices in the mesh are placed in their original positions, thus retrieving a mesh preserving the details present in the input point set.

Finally, in order to get an idea of the behaviour of this method, Figure 4.5 presents some results.

Spiraling Edge The Spiraling Edge proposal of Crossno and Angel [58] follows a similar approach to the Ball-Pivoting. This method assumes knowledge of the normals, as well as knowledge of the neighborhood of each point, so these computations are not considered part of the algorithm. The authors claim low computational cost in the execution of the method, but this is mainly due to these restrictive assumptions. It works by producing a local approximation of the surface by creating a star-shaped triangulation between a point and a subset of its nearest neighbors. This surface patch is incrementally extended by locally triangulating the neighbors of the points on the boundary of the current surface that have not yet been triangulated. The creation of new triangles on the boundary is performed in counterclockwise order around the normal of the points, starting from the first one and following the creation order. Boundary points require special treatment. As we can see, this method assumes locality on the points, as done by [28], meaning that they assume the surface to be sampled densely enough to be able to perform the reconstruction locally by projecting neighbors of a point onto its tangent plane.

Lower Dimensional Delaunay Another relevant method, also relying on the locality property, is the one presented by Gopi et al. [92]. It also presents guarantees for their defined sampling criterion that causes the obtained $\bar{S}$ to be homeomorphic to S. The algorithm presented in this paper is based on reconstructing the surface locally for each point in its tangent plane using a local 2D Delaunay triangulation. Moreover, it allows for surfaces with boundaries.

The δ-sampling definition introduced by the authors is based on the local curvature of the surface. Let the principal curvature k_v at a given point p and in a given direction v be defined as follows:

$$k_v = k_1, \tag{4.1}$$

k_1 and k_2 being the principal curvatures at that point, and θ the angle that v makes with the principal direction. With k_v, they define a closed contour C_p^δ on a surface spanned by the two points p_i and p_j along with their normals and local curvature values:

$$\forall p_j \in C_{p_i}^\delta,\ k_v \cdot arclength(p_i, p_j) = \delta, \tag{4.2}$$

where $arclength(p_i, p_j)$ is the Euclidean distance between the two points on S, and the direction v is defined by the projection of the vector $\overrightarrow{p_i p_j}$ onto the surface. Then, a given P is a δ-sampling of a surface S if every point $p_s \in S$ has a closest point $p_p \in P$, so that $p_p \in C_{p_s}^\delta$. Basically, this sampling condition requires the points to be closer in high curvature regions, while allowing them to be more spread out in low curvature parts.

Regarding the algorithm, it starts by computing the normals for each point in P, if they are not known beforehand. The procedure for computing and orienting these normals is similar to that used by Hoppe et al. [104] (despite its formulation being different). Once per-point normals are available, they compute the curvature. This is achieved by using the k-nearest neighbors of the points, and using the formula of [203]. Given a point p_i and one of its neighbors p_j, the normal curvature along the direction $\vec{v_i} = (\overrightarrow{p_i p_j} - (\overrightarrow{p_i p_j} \cdot n_i) n_i)$ in the tangent plane at p_i is defined as:

$$k_p^{v_i} = \frac{2(n_i n_j)\overrightarrow{p_i p_j}}{\|\overrightarrow{p_i p_j}\|^2}. \tag{4.3}$$

Taubin proposes to compute the curvature tensor as:

$$M_p = \sum_{i=1}^{k} u_{p_i}^i k_{p_i}^{v_i} \vec{v}\vec{v}^T, \tag{4.4}$$

where $u^i_{p_i} = \frac{a_{i+1}-a_{i-1}}{4\pi}$, a being the angles sorted in counterclockwise order, and v_i being expressed in terms of the principal directions defined by the local tangent plane. The minimum and maximum eigenvalues of this matrix represent the principal curvatures k_{min} and k_{max} of each point.

Using the previously computed curvatures, the candidate points around a given p_i are selected as being in the 2δ neighborhood of the point. The maximum ratio of distances between two points in the contour $C^{\delta}_{p_i}$ is the ratio of principal curvatures at point p_i. Being $dist(p_i, K_1(p_i))$ the distance from p_i to its closest neighbor, and the constant $n = \frac{2k_{max}}{k_{min}}$, the neighbors to take into account are those inside the sphere of radius $r = m_p \cdot Dist(p_i, K_1(p_i))$ around p_i. Points inside this neighborhood are further refined by computing the height values of these points in the tangent plane of p_i, and eliminating those whose value is greater than $\frac{\sqrt{1+4\delta^2}-1}{k_{min}}$, since these points are outside the tubular neighborhood of the surface around p_i.

Each of the candidate points are triangulated in the local tangent plane. This triangulation starts by sorting the angles (taking into account the quadrant they lie in) that the points form. Then, the points are taken in triplets, p_a, p_b and p_c, at each iteration, and are checked to see if a point p_b (middle one) can be a Delaunay neighbor of p_i in presence of p_a and p_c. All candidate Delaunay neighbors are stored in a list ordered by their radial angle with respect to the central vertex. Vertices p_a, p_b and p_c form a triangle if all (p_b, p_c), (p_c, p_a) and (p_a, p_b) are consecutive neighbors in their correspondent ordered lists.

Smooth Greedy As opposed to the methods proposed so far, Cohen-Steiner and Da [54] propose a method using the 3D Delaunay triangulation as a base. Assuming a smooth surface, it prioritizes minor variations on the curvature of the growing surface. By doing this, ambiguous decisions can be overcome by the advancing front coming from other directions. Thus, the method uses a unique advancing front that incrementally constructs the desired surface by running along the structure of the Delaunay triangulation.

To ensure maintaining a manifold mesh at all times, the authors divided the possibilities arising from adding a new triangle to the current surface into four. Given the new point p_i generating the candidate triangle, and the associated edge $e_{j,k}$ on the boundary of the current surface, the four cases are the following:

- **Extension:** p_i is not in the current surface.
- **Hole filling:** p_i is on the boundary, and both neighbors of p_i in the boundary are also endpoints of $e_{j,k}$.
- **Ear filling:** p_i is on the boundary, and only one of the neighbors of p_i in the

boundary is also an endpoint of $e_{j,k}$.

- **Gluing:** p_i is on the boundary, and no neighbor of p_i in the boundary is an endpoint of $e_{j,k}$.

Then, the selection of the next triangle to insert in the surface is twofold: first, choose a candidate (among all valid ones) for each of the edges forming the boundary of the current surface mesh, and second, choose a triangle from all those previously selected. The choice of candidate triangles for each edge is guided by the radius of their circumspheres, getting at each step the one minimizing it. This idea builds on the fact that smaller triangles are more likely to be part of the surface, under a dense and more or less regular sampling assumption. Sliver tetrahedra can introduce errors if only following this idea (e.g., surface foldovers), so a candidate triangle is discarded if its angle with the current surface is below a threshold (set to $5\pi/6$ in the paper). Starting with the triangle having a minimum circumradius, new triangles are inserted in the surface guided by the dihedral angle they make with the current surface, prioritizing small values. However, if these dihedral angles are below a threshold, their priority is guided by their circumradius $rad(t_i)$, being $priority(t_i) = 1/rad(t_i)$. On the other hand, if the dihedral angle is over the threshold, its priority is set to $priority(t_i) = -dihedral(\bar{S}, t_i)$, where $dihedral(\bar{S}, t_i)$ is the dihedral angle between the currently growing surface $\bar{S}$ and the candidate triangle t_i.

The authors further propose two variants of the algorithm dealing with more than one component (multiple surfaces), and with boundaries. For multiple components, the idea is simple: the process has to start again from another triangle formed by points that have not yet been visited. For boundaries, triangles having a large circumcircle (greater than a threshold) are discarded. Thus, boundaries are seen as undersampled parts of the surface, and large triangles are likely to fill an undersampled area. Notice that using a 3D Delaunay triangulation naturally eliminates the possibility of self-intersection problems on the growing mesh, as it can only intersect with itself at points or edges of the triangulation, not between triangles. In Figure 4.6 one can see some steps of this greedy procedure, showing how the method grows the surface in low curvature areas prior to facing the more curved ones.

To conclude, we have mentioned some of the more relevant greedy surface methods, but there are many other proposals that can be seen as small variants of the presented approaches, either using the Delaunay structure [129], or a data structure-free greedy procedure in space [142, 108, 140].

Medial Scaffold Transform Finally, the method by Chang et al. [48] is very different in essence from the rest of the methods presented in this category. In this case, the process

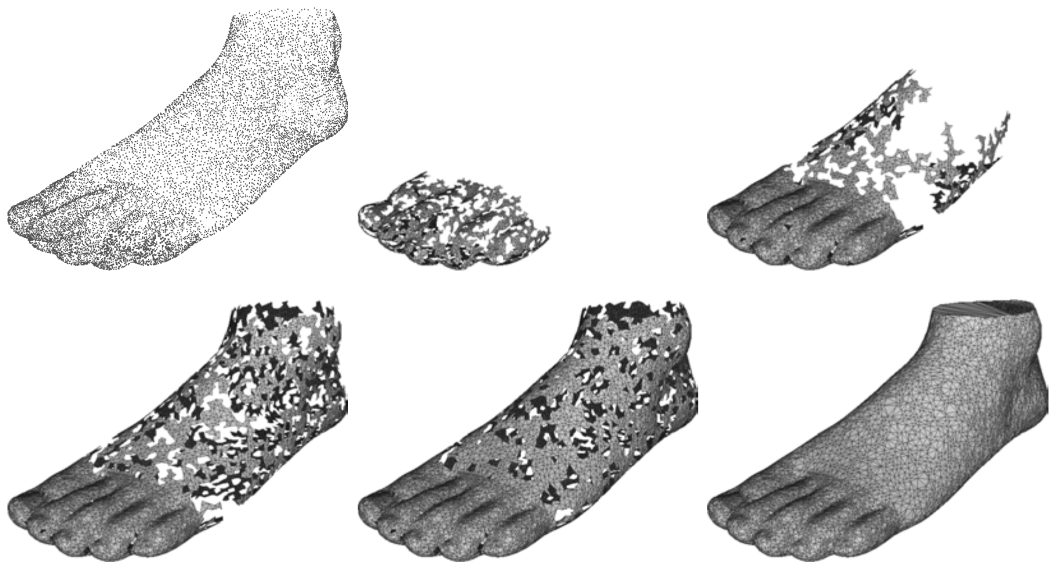


Figure 4.6: Various iterations of the Smooth Greedy algorithm [54]. Top-left corner shows the original point cloud, and then from left to right, top to bottom, different steps of the evolution of the greedy surface are presented.

is guided by a structure called the Medial Scaffold (MS(P)).

The Medial Scaffold is a graph representation of the Medial Axis of an object, defining the relations of how medial curves (edges in MS(P)) connect medial points (vertices in MS(P)). This graph can be computed directly from a point cloud, as suggested by Leymarie and Kimia [138]. The main idea is to see the point cloud as a deformation of the original S, where holes have been growing till only infinitesimal points remain. During this deformation, the MS(P) suffers a series of changes. Thus, the reconstruction process can be seen as an inverse transform of the MS(P), i.e., one can find the surface by *undoing* the transitions that lead to the current MS(P).

There are many transitions during the deformation process, but this paper only considers a type of transition, called the gap transform. The gap transform is based on the fact that removing a triangle patch from the surface generates a group of three sheets, together with a curve in the Medial Axis at their intersection. Inverting the process consists of finding these curves and their associated triplets of sheets, i.e., their representation in MS(P), and change them by a surface triangle. However, the order in which the gap transforms are computed is important.

For this reason, a two stage greedy approach is used. The set of curves are divided into two queues. The first queue is made up of curves that contain the circumcenter of the triangle that should recover by its inverse transformation, since these are the simple cases (high confidence of generating a correct triangle). The second queue contains the rest of the curves, which can lead to ambiguities. For ordering the curves without local connectivity information (1st queue), the algorithm favors compact triangles and triangles smaller than the length of their associated curves. The length of the curve is its shortest path when divided by the circumcenter of its generating triangle. On the other hand, curves having local context information are divided by whether they can share an edge or a vertex with an already reconstructed triangle. If they can share an edge, they are likely to be correct if the dihedral angle between them is small. If the triangle can share a vertex with the already reconstructed triangles, the algorithm aims to preserve one-ring (fan) ordering around a triangle.

4.4.1.3 Integration

A strong prior is required by the algorithms in this section, which is knowing some local connectivity about the input points at the time of the capture. That is, these kinds of methods assume a range scanner to be able to provide a 2 dimensional array of ranges as output. Scanning methodologies fulfilling this property could be, for example, a computer vision 3D reconstruction pipeline, where we know the local connectivity between points in the 2D image plane, or most time-of-flight cameras.

Zipper The representative method of this category is the Zippering technique proposed by Turk and Levoy [204]. In this method, the local connectivity information of a local image range scan is used to define the surface locally. Then, assuming that we have already registered the local scans in a global coordinate frame, the problem is reduced to merging local triangulations together in a single surface by triangulating overlapping areas again to achieve continuity. A new range scan is added at each iteration using the following steps:

1. Remove overlapping portions of the mesh: Remove the redundant triangles on the boundary of the first mesh, and then from the second. A triangle is redundant if its three vertices have correspondent vertices on the other mesh within a tolerance distance d.

2. Clip a mesh against the other: After removing redundant triangles, the two meshes slightly overlap. If the process was taking place in 2D, it would be as follows: take the set of points P^i that are intersection points between edges of mesh $\bar{S}_a$ and mesh $\bar{S}_b$. Triangles from mesh $\bar{S}_b$ need to be split once for each vertex, while triangles from mesh $\bar{S}_a$ loose a part of them, because it falls under the boundary of $\bar{S}_b$. Since in 3D these edges from borders of mesh $\bar{S}_a$ and $\bar{S}_b$ do not need to intersect, an augmentation of the border of $\bar{S}_b$ is performed by adding a collection of four triangles that are roughly perpendicular to the boundary of $\bar{S}_b$ (two triangles over

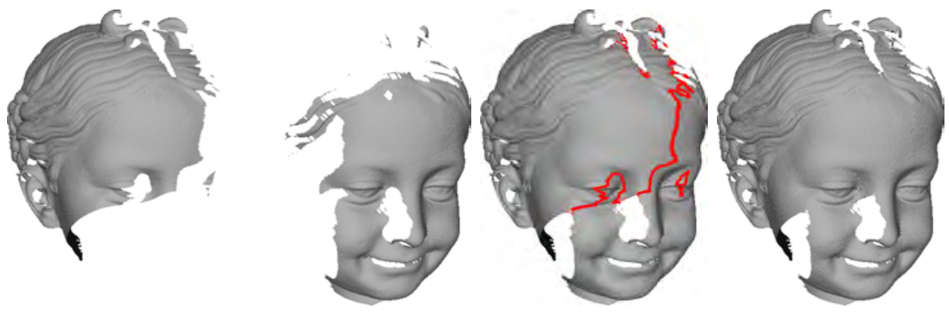


Figure 4.7: Zippered meshes, using the implementation of the variant proposed by Marras et al. [154]. The first pair of figures shows two separated range scans. The next pair of images shows the merged model, the first denoting in red the border where the zippering took place.

the border and two below the border, forming a quadrilateral for each pair). This augmentation can be seen as forming a *wall* perpendicular to the boundary of $\bar{S}_b$, so the procedure described in the plane can be applied in 3D.

3. Remove small triangles introduced during clipping: This part pretends to improve the quality of the resulting mesh, since the clipping step generates small triangles which may not be desired depending on the application of the surface. The method consists of eliminating all the triangles having an altitude greater than a threshold by deleting one of the vertices of the triangle and all the triangles that shared it. Then, a constrained triangulation is used to fill the generated holes.

A sample of the behaviour of a variant of the method, presented by Marras et al. [154], can be seen in Figure 4.7.

Venn Zipper Also defined for range images, is the method devised by Soucy and Laurendeau [199]. In this case, different parts of the model are estimated by a set of triangulations modeling each canonical set of the Venn diagram of the different views. The Venn diagram of the set of views represents the parts/strips of the object that are viewed from different views at the same time. Then, the integrated surface model is built from these local surface models describing each canonical subset of the Venn diagram.

The Venn diagram of a set of range views can be computed by obtaining the common surface parts between each pair of views. A canonical subset of the Venn diagram is the one that has the points that are visible in a particular combination of range views. Thus, we need to detect views for each point in the range, which other range scans have also sampled (approximately). Since the same point is never sampled exactly in two different range scans, this *matching* between points is approximated. A point in one image is considered as sampled in another image if the following two tests apply:

- Surface Neighborhood Test: The point must be located near a surface patch sampled by the second image. For each of the surface patches in the second image, projected to 3D, an uncertainty zone based on the error of the measuring device is defined, and if a point falls inside this zone, it is marked as visible in the other view.

- Surface Visibility Test: A point in the first range image must be visible in the second view. This is done by using the dot product on the surface normals.

After applying the previous two tests, a region growing approach is used to refine the discontinuities between canonical subsets of the Venn diagram.

The constructed Venn diagram is traversed from top to bottom, the top being the canonical subset having the largest number of views. Then, a parametric grid is built from each canonical set, having an orientation equal to the average of the orientations of the views in the set. All the information from the different views is merged and averaged in this parametric grid. Once we have a parametric grid from each canonical subset, an integrated model is built by re-parametrizing in a global grid again, so the final integrated model is a 2.5D triangulation defined in a single reference frame, coinciding with that of the first image.

4.4.2 Volume Oriented

Since these methods rely on the volume, an initial spatial decomposition in unjoined cells is used as a base by the methods in this category. Although not exclusive, this decomposition is usually the 3D Delaunay triangulation. The proposed classification for volume oriented methods is as follows:

- **In/Out Separation:** Cells in the structure are marked as belonging to the inside/outside of the surface according to some properties. Then, as in the Triangle Selection case, cells belonging to the same part of the volume are joined incrementally to form two final sub-volumes from which we can extract the surface mesh as the interface between them.

- **Sculpting:** Cells are sculptured (i.e., deleted) from the initial spatial decomposition from outside-inside following an order defined by some priority rules.

- **Graph Partitioning:** Similar to the In/Out Separation methodology, but here the initial spatial decomposition is transformed to a graph representation, having weights defined by some properties. Having this weighted graph representation, a graph partitioning technique is used to separate the two inside/outside volumes.

4.4.2.1 In/Out Separation

Methods in this section are related to those in Section 4.4.1.1 in that they follow the same idea of defining a set of cells following some property, and then join them iteratively. Nevertheless, in this case, methods seek two sets of cells defining the inside and the outside of the object. Basically, cells being more likely to be part of one set or the other are defined, and then two complementary sets of inside/outside cells are created from them.

Power Crust The first and most representative method in this class, given its many citations, is the Power Crust method proposed by Amenta et al. [13]. The method first builds an approximation of the medial axis transform defined by P and then uses its inverse transformation to get the surface.

The MAT(P) is approximated using the poles, and more precisely the correspondent polar balls $B(q_i)$, which are balls centered at q_i and touching nearby points. The radius $rad(q_i)$ defines weights on the poles that are used to build a power diagram. Polar balls corresponding to poles inside the surface are called inner polar balls, and polar balls corresponding to poles outside the surface are called outer polar balls. The Power Crust is the boundary between the cells in the power diagram belonging to the inner poles and the cells in the power diagram corresponding to the outer poles. It has been demonstrated that inner and outer polar balls should intersect slightly, if at all. Power Crust interpolates the input samples which lie on the surface of the union of the inner and outer polar balls. Furthermore, the algorithm provides a secondary output, which is an approximation of the medial axis. This approximated medial axis is called power shape, and is retrieved from the regular triangulation, the dual graph of the power diagram. It is formed by the simplices connecting poles whose cells are adjacent in the power diagram.

Once the power diagram is built, the poles have to be classified as being inside (inner) or outside (outer) the object. This procedure starts from the poles adjacent to the bounding box $bbox(P)$. Then, this initial labelling is propagated based on the angle of intersection between polar balls: if they intersect deeply, the balls are part of the same subgroup, while two balls belonging to different groups should intersect slightly. During a greedy procedure, each ball has two associated values *in* and *out*, lying between 0 and 1. These values denote the confidence of the pole for being inside or outside respectively. The algorithm starts by labeling the poles adjacent to bounding box points with an *out* value of 1 and an *in* value of 0, and in/out values of 0 to the other. Then the poles are visited and labeled following a priority queue, whose priority is defined by in/out values in the following way:

- If only one of the two values is greater than zero, then its priority is the non-zero value.

- Otherwise, if both values are greater than zero, that means the labeling for this pole is confusing, and is added to the queue with a priority $|in - out| - 1$.

At each iteration, the top element of the queue is removed and its label is fixed to the greater of the two values *in*/*out*, updating the weights of the remaining unlabeled poles.

In order to equip the method with resilience to noise, the authors propose discarding some poles according to a measure of how elongated the Voronoi cells are. A lower bound of the $lfs(p)$ is heuristically estimated using the distances from the point to the poles and some user parameters. Then, the poles having a distance to their corresponding sample below the computed bound are not taken into account. Using the subset of poles over the threshold, they ensure that only *skinny* and elongated Voronoi cells are retained. Furthermore, by discarding both poles of a sample when either of them does not pass the previously mentioned test, the method acquires the ability to recover sharp features (even if not sampled).

They also propose extensions to the algorithm. The first one deals with holes (boundaries) in the surface, by using the fact that, in such places, inner and outer polar balls intersect deeply. The second one is an offset surface computation from the MAT(P), by adding/subtracting a constant value from each of the polar balls (e.g., subtracting in interior polar balls and adding in exterior polar balls to get an inside offset surface).

Notice that the output of this method are not necessarily triangles, but polygonal faces (that can be easily triangulated). Also, not all the input points have to be necessarily part of the reconstructed surface, and some new vertices may appear. To conclude, the application example in Figure 4.8(b) demonstrates the validity of the method.

Robust Cocone A method following a similar strategy, but providing theoretical guaranties in case of noisy data is the Robust Cocone method presented by Dey and Goswami [65]. Here, a set of large Delaunay balls, resembling the polar balls in Power Crust, are used to devise the surface. They prove that, given the assumed noise model, the union of inner/outer Delaunay balls is homeomorphic to the underlying surface. The noise model is described theoretically, and is based on the lfs sampling criterion. Given a point set P, it follows a (ϵ, k)-sample of the original surface S if:

- The sample without noise P^I (I for *Ideal*) is an ϵ-sample of S, that is, $dist(p^I_i, P^I) \leq \epsilon \cdot dist(p^I_i, \mathrm{MA}(P))$ for each point $p^I_i \in P^I$. This property assures dense sampling.

- $\|p_i - p^I_i\| \leq \epsilon^2 \cdot lfs(p_i)$. This assures that the sampled points are close to the ideal surface/sampling.

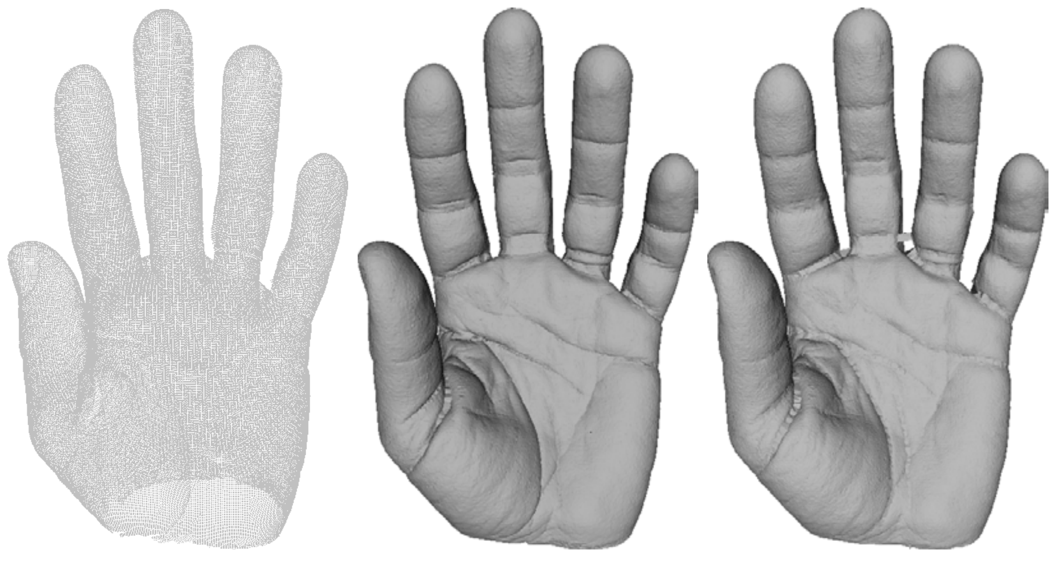

(a) Point set (b) Power Crust (c) Robust Cocone

Figure 4.8: Sample of both methods presented inside the In/Out Separation category. Note how there are small differences between the Power Crust and Robust Cocone results (loss of small details and a bridge between fingers for Robust Cocone).

- $\|p_i - p_j\| \leq \epsilon^2 \cdot lfs(p_i^I)$ for any two points p_i, p_j in P where p_j is the k-nearest sample point to p_i. This property forces the sampling to be locally uniform.

As in the Power Crust algorithm, the overlap (or intersection) between balls is used to separate inside/outside tetrahedra. The algorithm chooses a Delaunay ball, and expands the set (inner or outer) by iteratively adding the neighboring ones that intersect with the already constructed set at an angle larger than a given threshold. The boundary of the inner/outer set of balls is the surface. Despite the procedure explained, the final implementation they propose uses the set of balls to filter P to those on the boundary of the balls, and then apply the Tight Cocone algorithm [64] (described in Section 4.4.2.2).

As observed in Figure 4.8, the behaviour of both Power Crust and Robust Cocone is similar.

4.4.2.2 Sculpting

The algorithms falling into this section cast the surface reconstruction problem as a sculpting process. The initial volume (normally the Delaunay triangulation of P) is iteratively refined from the outside to the inside to reveal the final surface/volume of the object.

Boissonnat's Sculpting The first approach in this category corresponds to the second proposal in the previously mentioned article by Boissonnat [28]. Starting from the 3D

Delaunay triangulation of the input points, tetrahedra are eliminated sequentially until all points are on the exterior of the volume. This procedure is restricted to reconstruct polyhedrons of genus 0 (objects without holes).

The elimination procedure is based on maintaining the definition of a polygon for the current volume after each tetrahedron is removed. In order to preserve this property, and defining $\bar{S}_B$ as the boundary of the polyhedral shape, the only tetrahedra that can be eliminated are the ones with one face, three edges and three points on $\bar{S}_B$, or alternatively those with two faces, five edges, and four points on $\bar{S}_B$. The order in which tetrahedra are eliminated is guided by a value assigned to each tetrahedron with at least one face on $\bar{S}_B$. This value is defined as the maximum distance from the faces of the tetrahedra on $\bar{S}_B$, and the circumsphere of the tetrahedra. The algorithm ends when all the input points are on $\bar{S}_B$ or the maximum value of the heuristic applied to the current interior tetrahedra does not decrease with the elimination of the next tetrahedra.

Tight Cocone On the other hand, the Tight Cocone method described by Dey and Goswami [64], as its name implies, is an extension of the Cocone algorithm to deal with watertight surfaces. Even though the Cocone algorithm also assumed watertight surfaces, the initial implementation produces undesirable small holes in the final surface located in undersampled regions.

The method starts by constructing a first approximation of the surface using the Cocone method. A labeling algorithm on the 3D Delaunay triangulation is used to decide which tetrahedra are outside the surface and which are inside. Then, a peeling algorithm is needed to remove the tetrahedra marked as outer, as well as some other tetrahedra that were unclearly labeled. The labeling is based on the definition of good points, which are points having incident triangles in the Cocone forming a topological disk.

The labeling maintains a stack of pairs containing a good point and a tetrahedron marked as *out*. Using the separation that an umbrella (topological disk of triangles) generates of inside and outside, we can iteratively mark tetrahedra and update the stack using adjacency relations of the tetrahedra already marked *out*, taking into account that an umbrella cannot be crossed. All marked tetrahedra are incident to at least one good sample point, while tetrahedra not having any good points are called *poor* tetrahedra.

After labeling, the peeling process consists of removing the outer tetrahedra in order to reveal the inside volume of the object. It maintains a stack of surface triangles that form the boundary of the union of peeled tetrahedra. At each step, a triangle t_i from this stack is popped out. Since it is on the stack, one of its associated tetrahedra is already peeled, and we have to check the state of the other. If the other one is also already peeled, the triangle separates two tetrahedra on the outside, so they are not part of the surface. Otherwise, there are three possibilities depending on the properties of the current non-peeled tetrahedron tet_i:

- tet_i is not *poor* and is marked *in*: Insert it in the output list.
- tet_i is marked *out*: The peeling must move inside this tetrahedron through t_i, so the stack is updated with the three triangles different from t_i of tet_i.
- tet_i is marked as *poor*: Also walk inside tet_i, as in the previous case, if t_i is not the smallest triangle in tet_i. This last check is performed because the locality idea of Dey and Goswami [64] states that the smallest triangle of a tetrahedron of an undersampled region separates the interior from the exterior of the object.

At the end of the peeling stage, the boundary of the inner volume is retained as the reconstructed surface. The behaviour of this algorithm can be seen in Figure 4.9 (b).

Convection The convection approach presented by Chaine [47] also follows a sculpting methodology. The algorithm tries to implement the convection method of Zhao et al. [216], which is based on modelling how a curve is deformed when attracted by its distance to the input points. The difference with the method in Zhao et al. [216] is that, instead of using and moving an implicit representation, the authors use a discrete version level set.

More precisely, they control the evolution of a pseudo-surface embedded in the 3D Delaunay triangulation. The pseudo-surface is made of half-facets, which are facets with an orientation. Also, a pseudo-surface is not a simplicial complex, since two adjacent half-facets can share more than one edge. Apart from the half-facet concept, they also define the concept of half-balls, which are the part of the smallest enclosing ball of a facet falling in the half-space defined by the half-facet. They also extend the notion of Gabriel to half-facets, being those with half their ball empty. The pseudo surface obtained is oriented inwards, its half-facets are part of the Delaunay triangulation of the input points, and they also meet the Gabriel property.

The algorithm starts with the pseudo-surface being initialized to CH(P), with the half-facets oriented inward. Then, it evolves inside the Delaunay triangulation by sculpting some tetrahedra. The surface shrinks inward by making the area it encloses decrease. This evolution stops when all the half-facets meet the Gabriel property. At each step, if a half-facet does not meet the Gabriel property, there may be two possibilities:

- The half-facet corresponds to a facet in the pseudo-surface with a different orientation: This means that the pseudo-surface has met itself during convection. If surfaces without boundaries are assumed, these kinds of facets are eliminated directly. On the other hand, under the assumption of bounded surfaces, these cases are acceptable. When collapsing a pair of coincident half-facets, 8 possibilities must be taken into account (see original text for details).

- Otherwise, replace the half-facet with the three half-facets it is hiding.

This method stops before reaching all the points if pockets are part of the surface, even if they are sampled enough. Pockets are important concavities larger than the hidden half-balls that interface between the cavity and the outside. To solve the early end of the algorithm in these cases, a half-facet that meets the Gabriel property can be further shrunk if its size is not coherent with the local 3D density of the sampling. The density around a point is approximated, in this case, using the distance to its four nearest points.

Wrap Another representative example for this category can be found in the Wrap algorithm of Edelsbrunner [71, 70]. As in the previously mentioned method of Giesen and John [88], the concept of flow is used, but in this case to define a relation between simplices on a Delaunay triangulation. The flow distance function, defined by the points P, has only one direction of steepest ascent at each possible $p \in \mathbb{R}^3$, except at critical points. This property is used to define the flow relation, which is an acyclic sequence of contractions on Delaunay simplices following the flow induced by the point set. In this sense, a relation involving a triangle t_i and its two associated tetrahedra tet_i and tet_j exist if there is a point $p \in \mathbb{R}^3$ on t_i that passes from tet_i to tet_j following the flow.

The flow defines the relations of *ancestor* and *descendant* between simplices. Using these notions, the simplices in the Delaunay triangulation can be classified in three types:

- Centered: Simplices containing critical points in its interior. They do not have ancestors nor descendants.

- Confident: The simplex is not centered and its affine hull intersect is a dual Voronoi face.

- Equivocal: The affine hull of the simplex intersect is a dual Voronoi face.

A sink is a simplex in the Delaunay triangulation of the input points that do not have a predecessor in the flow relation. By definition, sinks are centered simplices, including a simplex at infinity, to account for the unbounded side of the convex hull of the set of points. Each of these sink simplices (finite or unbounded) generate a sequence of ancestors, and the set of all the ancestors from all the finite (or equivalently, the infinite) simplices form a volume, whose boundary is the required $\bar{S}$. In order to obtain this surface, the algorithm iteratively collapses tetrahedra, starting with the CH(P), and following the ancestor/predecessor relations between simplices. A collapse shrinks the current Delaunay subcomplex, and only 6 cases can occur (see original text for details).

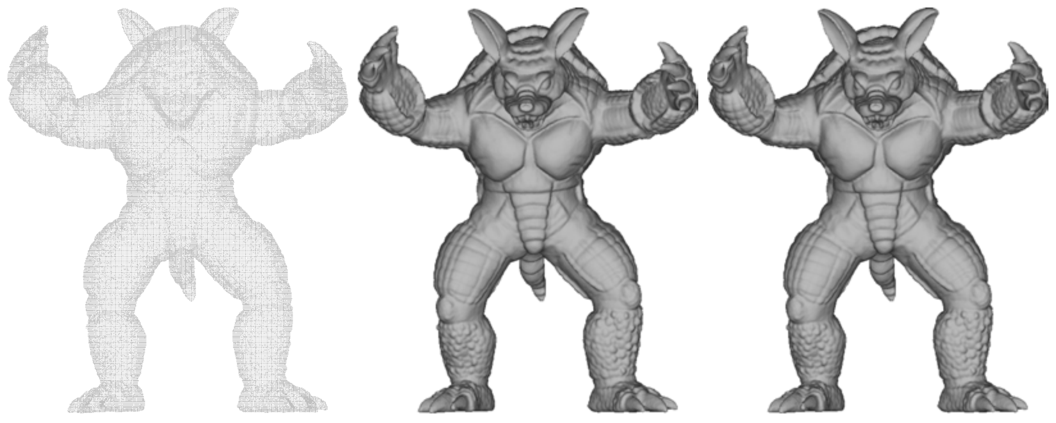


Figure 4.9: Examples of the behaviour of the Tight Cocone and Peel algorithms when applied to the Armadillo dataset.

Contrary to most algorithms, the order of the sequence of collapses does not matter, and the algorithm always gets the same final result. This collapse maintains the homotopy type of the Delaunay complex, so the retrieved surface is homeomorphic to a sphere. To allow for other topologies, a simplex removing operation is added to the algorithm, and then collapses are not only computed for the infinite sink, but also for other finite sinks.

Peel Another approach is the Peel algorithm presented by Dey et al. [66]. This method is based on iteratively deleting tetrahedra from an α-complex for a sufficiently small α (proposed value for α is six times the largest nearest neighbor distance among the given points). The peeling process consists of removing edges and their two associated triangles, without deleting any vertices. While there remain tetrahedra to be peeled, the algorithm removes them in a top-down way. It is demonstrated in the article that following a top-down approach leads to a surface $\bar{S}$ that is isotopy equivalent to the unknown surface S. The algorithm it is also proven to give an isotopic reconstruction in the case of bounded surfaces. Figure 4.9(c) shows an example of the application of this method.

4.4.2.3 Graph Partitioning

Algorithms in this section build a graph structure out of the point cloud (usually based on Delaunay/Voronoi), associates some weights to its edges, and applies a graph partitioning algorithm in order to find the separation of tetrahedra being inside or outside the object.

Eigencrust The first algorithm using graph partitioning techniques is the Eigencrust of Kolluri et al. [125]. This noise resilient algorithm reconstructs a watertight surface from a point cloud using spectral graph partitioning on a graph embedded in the Delaunay triangulation of the input point cloud. The spectral partition is used to label each tetrahedron as being inside or outside the shape of the object. As usual, the surface is found as the interface between *in* and *out* tetrahedra.

The labeling is performed in two steps. First, the algorithm only takes into account the poles computed from the Voronoi diagram of the input point set. The poles are used as vertices of the so-called pole graph. Poles q_i^+ and q_i^- from the same sample share an edge in graph G_1. Also, if there is an edge in de Delaunay triangulation that joins vertex p_i, having poles q_i^+ and q_i^-, and p_j, having poles q_j^+ and q_j^-, then the edges $\mathsf{e}(q_i^+, q_j^+)$, $\mathsf{e}(q_i^+, q_j^-)$, $\mathsf{e}(q_i^-, q_j^+)$ and $\mathsf{e}(q_i^-, q_j^-)$ are also in G_1. Each of these edges has a weight $w(\mathsf{e}(q_i, q_j))$ proportional to the angle of intersection of the circumspheres described by each of the poles:

$$w(\mathsf{e}(q_i, q_j)) = -e^{4+4\cos(iangle(B(q_i),B(q_j))}, \tag{4.5}$$

where $iangle(x, y)$ is this intersection angle. Poles likely to lie on the same side of the surface have a large positive weight, while their weight is small when they barely intersect. When the angle of intersection is 0 (no intersection), the edge is removed from the graph. Note that this heuristic (angle of intersection between polar balls) is the same used in [13] (Power Crust). Furthermore, nodes known to be outside the model (lying on a bounding box for example) can be collapsed into a *supernode* representing all of them.

Given graph G_1, the weighted adjacency matrix L, with negative weights, is built. Then, the eigenvector associated with the smallest eigenvalue λ of the following system is computed:

$$Lx = \lambda Dx, \tag{4.6}$$

where matrix D corresponds to the diagonal of L. Each component of x corresponds to a pole, i.e., to a tetrahedron. Tetrahedra corresponding to components in x having the same sign as the outer supernode are labeled outside, while those having an inverse sign in x are labeled inside.

The remaining tetrahedra are labeled in a second step, by computing a new graph G_2. Each unlabeled tetrahedron forms a node in the graph, and two supernodes representing the tetrahedra already labeled as *in* or *out* respectively are created. Then, an edge is added for each node sharing a face in the Delaunay triangulation, and an edge to the corresponding supernode is also added if an unlabeled tetrahedron shares a face with an already labeled one. Weights between tetrahedra in this graph follow the heuristic of enforcing *aspect ratio* of the joining triangular faces, encouraging regularity on the triangles of the output surface. Graph G_2 has only a negative weight, connecting the two *in*/*out* supernodes. Once constructed, it is partitioned using the same procedure as G_1.

Optionally, the authors also propose an alternative to this second labeling step, consisting of using the labeling of the first partition to label power cells, substituting the Power Crust labeling algorithm. Power Crust can be directly extracted from this labeling. The resulting surface may be non-manifold, so a relabeling step that takes into account manifoldness should be needed. It is worth noticing that this algorithm presents good results in the case of having a large number of outliers as part of the input points.

Graph Cuts Stereo Another example in this section is that of Labatut et al. [132], based on using some additional information generated when creating the input point cloud. In this case, the input point cloud is assumed to be extracted from a computer vision multiple-view stereo 3D reconstruction pipeline, that is, from the matching of image keypoints across images in a calibrated system. A Delaunay triangulation is built from this point set, and its dual, the Voronoi diagram, is used to define a graph built on top of it. The surface is extracted from this graph representation using an S-T cut (source-sink cut) method [37] whose energy minimization takes into account the area of the surface, its visibility and its photoconsistency.

When building the input point set, keypoints in the images are matched and a discrete (point based) reconstruction of the object/scene is retrieved. Then, a Delaunay triangulation is built incrementally, adding one point at a time. When inserting a point into the triangulation, its nearest neighbor is selected. The maximum reprojection error between the two 3D points is computed, and if this value is above a threshold, this point is considered a new point and is added to the triangulation. However, if the error is below that threshold, the point is considered to be the same and, instead of adding the new point, the position of the one already in the triangulation is updated. This procedure provides a cloud where each point has a number of views in which it has been seen. The graph built on the Delaunay triangulation of the input points has a node representing each tetrahedron, while an edge is added to the graph if the two tetrahedra representing the nodes share a triangular face. By seeing this definition, one can observe that the nodes and edges forming this graph are equivalent to the nodes and edges in the Voronoi diagram. Furthermore, two special nodes are added: the source and the sink. These two nodes do not correspond to specific positions in space, and each node in the graph has an edge that joins it with each of them.

Using this graph representation, the idea is to minimize the following energy functional using the S-T procedure defined in Boykov and Kolmogorov [36]:

$$E(S) = E_{vis}(S) + \lambda_{photo} E_{photo}(S) + \lambda_{area} E_{area}(S), \tag{4.7}$$

where λ_{photo} and λ_{area} are positive weights. Regarding the terms of the energy, $E_{vis}(S)$

accounts for visibility. Each center of each view (i.e., camera position) in the initial capturing system form a segment called line-of-sight ($LoS(p_i)$) with the points that have been seen from that position. Now, weights on the graph are set according to the intersections between these segments and the Delaunay triangulation. Taking the $LoS(p_i)$ of each point, an edge to the sink with weight λ_{in} is added for the tetrahedra after the point p_i and following the direction of $LoS(p_i)$. On the other hand, a λ_∞ weight is added to the edge joining the tetrahedra where the camera's viewpoint is located and the source. Finally, a λ_{out} weight is assigned to the triangles crossed by the ray (from inside to outside).

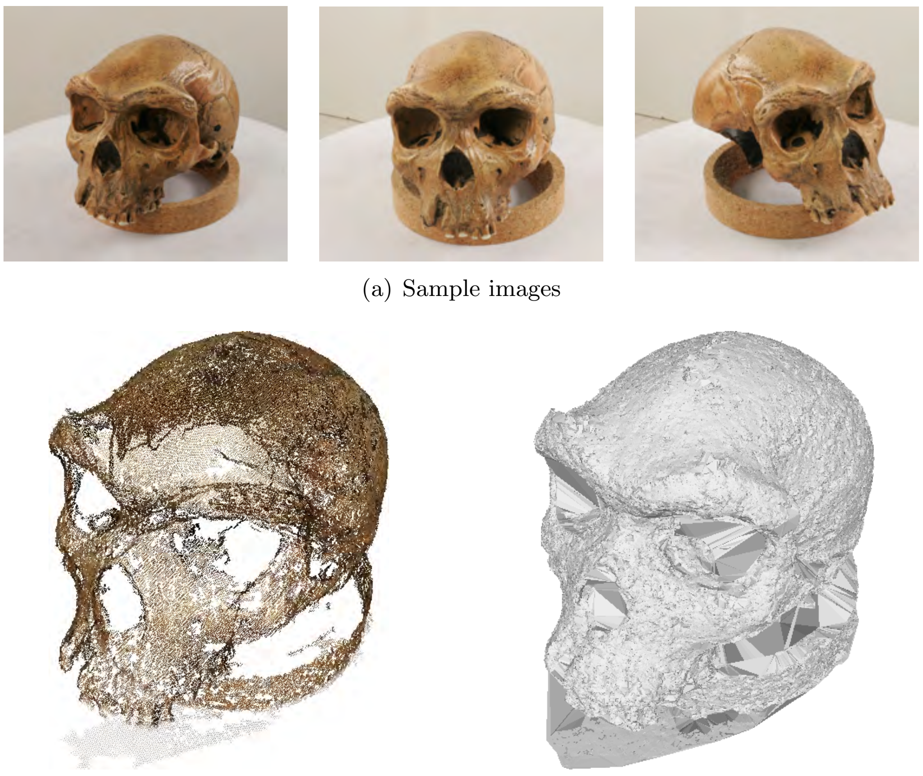
Next, the E_{photo} term accounts for photoconsistency of the triangles on the surface. This photoconsistency for each triangle is only computed in the views from which its three vertices were reconstructed. For each directed pair of tetrahedra (taking into account the normal of the triangle) a weight $w_{photo}(\mathsf{e}(i,j)) = \rho$ is added if the $LoS(p_i)$ and n_{t_i} of the triangle fulfill that $LoS(p_i) \cdot n_{t_i} > 0$, and where ρ represents the photoconsistency measure accross images. Any photoconsistency measure could be used, in the original paper they suggest the mean of the color variance of the pixels falling inside the projected triangle.

Finally, the $E_{area}(S)$ term enforces a minimum surface. This is achieved by adding for each edge of the graph, a weight $w_{area}(\mathsf{e}(i,j))$ with the area of the triangle it represents. It is worth noticing their proposal to include infinite vertices of the Delaunay triangulation and their associated tetrahedra into the minimization. By taking them into account, bounded surfaces are reconstructed even if the algorithm has a clear volumetric formulation. Additionally, it is mentioned in a later paper [99] that the photoconsistency is not accurate under dense sampling, since, under high sampling, triangles become small and the small texture patch contained in them is not informative enough. Finally, an extension to preserve weakly supported surfaces, seen at glancing angles from the cameras, is presented by Jancosek and Pajdla [112].

Graph Cuts Range The last method in this section is also a method from Labatut et al. [133], basically consisting of an extension of the previous algorithm to work directly on range scans. Since it still relies on lines-of-sight, these datasets must also contain the position from where each point was captured.

Now, in Equation (4.7), the visibility term has changed slightly and the photoconsistency term has been replaced by another term accounting for surface quality. In this case, the visibility constraint is relaxed by a parameter w that modifies the weights of the edges. First, the final tetrahedron along $LoS(p_i)$ is not the first after the end of the line-of-sight, but is the one at 3β distance from it, where β is a user's parameter. Also, both inside and outside weights of the intersected faces are smoothed out using β:

$$w_{softvis}(\mathsf{e}(tet_i, tet_j)) = 1 - e^{-d^2/2\beta^2}, \tag{4.8}$$



(a) Sample images

(b) Point set

(c) Graph Cuts Surface

Figure 4.10: Graph Cuts Stereo method [132]. First row shows a sample of 3 images from a total of 24 composing the SkullA dataset. The second row contains the point set and surface as obtained by the Graph Cuts Stereo method.

d being the distance between the intersection point of the current facet with the $LoS(p_i)$ and the input point p_i. Changing the β parameter is a means to control the smoothness of the resulting mesh. This change takes into account that range scans are very dense but also noisy, and consequently the resulting surfaces using the original visibility term get very bumpy. This is due to the mislabeling of interior tetrahedra produced by having a dense sampling with few lines-of-sight per sample. Note that this does not happen in the previous case when using computer vision features, as they are seen from more than one view-point and thus generate more than a single line-of-sight per point. On the other hand, the surface quality term aims at finding good quality triangles for the final surface. As in many other approaches (β-skeleton, Power Crust, etc.), this term aims at favoring the spheres bounding a tetrahedra to not intersect deeply at the shared face t. A weight that takes into account the angles γ_1 and γ_2 that the circumspheres of the 2 tetrahedra sharing a face make with the plane defined by that triangle is added to the graph for each edge:

$$w_{qual}(\mathsf{e}(tet_i, tet_j))) = 1 - \min(\cos(\gamma_1), \cos(\gamma_2)). \tag{4.9}$$

As noticed by the authors, the resulting surface does not have to be smooth nor visually pleasant. For this reason, they suggest to using a Laplacian smoothing on the resulting mesh to alleviate these problems. As previously mentioned, this recommendation extends to all the methods in Section 4.4.

4.5 Approximation-based Methods

Again, methods falling in this category are further subdivided according to common methodology. Nevertheless, and in contrary to the interpolation-based methods, the underlying surface representation the algorithms follow is not always the explicit one. Since we have relaxed the restriction of the final surface passing through the input points, most of the approximation-based algorithms use an implicit formulation to get the surface, and then gather the surface as a triangle mesh using an isosurface extractor method. Despite the predomination of implicit based algorithms, there are still some methods using an explicit formulation of the surface.

Approximation algorithms have been classified into the following subgroups:

- **Tangent Planes:** Compute signed distance functions considering per-point normals as tangent planes to the unknown surface.

- **Unsigned Distance:** Works with an unsigned version of the implicit distance to the surface as a base. In these cases, the problem consists of finding heuristics on how to extract the surface from this representation.

- **Radial Basis Functions:** Solve for an implicit signed distance function using the Radial Basis Function interpolation technique.
- **Moving Least Squares:** The implicit formulation of Moving Least Squares allows the definition of the surface as the zero level set.
- **Deformable Surfaces:** An initial rough guess of the surface is iteratively deformed towards the input points.
- **Gradient Enforcement:** Uses input points and associated normals as samples of a gradient field, and try to enforce these gradients in the final indicator/implicit function describing the surface.
- **Integration:** As in the Integration subgroup of the interpolation-based methods, local connectivity knowledge between samples is assumed or computed, and individual contributions are blended together in a global frame.
- **Local Primitives:** Comprises the methods working in a local view. Small primitives are computed locally around input points, and then used to build a global representation of the object. Nevertheless, the creation of the final surface also follows local procedures by using the close vicinity of each surface primitive at a time and using an explicit approach.

In Table 4.4 we can find the most relevant examples for each of the described groups. It is worth noting that given the approximate nature of the surface, theoretical guaranties are hard to define (note the very few methods having them). Nevertheless, another column has been added to the table indicating whether the surface representation the algorithm used is either implicit (I) or explicit (E). Recall that this column was not needed in Table 4.3 of interpolation-based methods, since they all use an explicit surface formulation. A detailed description of each category and the methods conforming it is provided in the following sections.

4.5.1 Tangent Planes

Tangent Planes methods use normals at input points (not necessarily required to be known) as a notion of where the surface should be to approximate the distance from any given point in $\mathbb{R}^3$ to $\tilde{S}$. Thus, they use the normals to generate an implicit function whose zero level set approximates the surface.

Hoppe's Method The method of Hoppe et al. [104], known for being the first to define the problem at hand as the *surface reconstruction problem from an unorganized point* *cloud*, is the first in this category. The method is based on using tangent planes at each of the input points. Having these planes as the approximation of the surface, the signed distance from a point $p^{\mathbb{R}^3}$ to the surface is computed as the distance to its nearest tangent plane. In addition to proposing a reliable computation of a tangent plane (normal) for each point, this paper also proposes a method to correctly orient the normals across the surface. This last part is crucial in obtaining a signed distance function, and has been intensively used in the literature.

Tangent planes are represented by a center c_i and a normal n_i. These two elements are determined using the $K(p_i)$, being the number of neighbors k a user parameter as usual. The center c_i corresponds with the centroid of the $K(p_i)$, while the n_i is retrieved by means of Principal Components Analysis (PCA). The normal is computed as the minimum eigenvalue extracted from the covariance matrix m created using the neighborhood:

$$m = \sum_{p_k \in K(p_i)} (p_k - c_i)(p_k - c_i)^T. \tag{4.10}$$

Once the planes are computed, a consistent orientation is required in order to be able to compute a signed distance function to them. The base structure for the normal orientation is the Riemmanian graph, an extended MST between centers of the tangent planes that contains an edge between two centers of planes if either $c_i \in K(c_j)$ or $c_j \in K(c_i)$. Then, a cost $w(\mathsf{e}(c_i, c_j)) = 1 - dihedral(n_i, n_j)$ is assigned to all the edges, and a traversal following the MST of the resulting graph is performed. During traversal, if $n_i \cdot n_j < 0$, then n_j is replaced by $-n_j$.

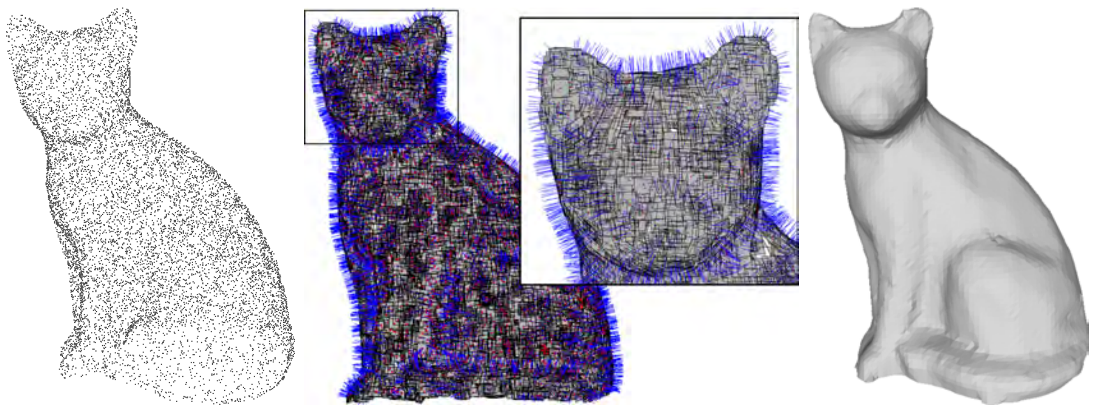
Once all the planes are oriented, the signed distance function for an arbitrary point $p_x \in \mathbb{R}^3$ is defined as:

$$I(p_x) = (p_x - c_i) \cdot n_i, \tag{4.11}$$

where c_i and n_i correspond to the closest tangent plane to p_x, computed as the one having its center c_i closest to p_x. This distance function is evaluated on each of the voxels v_i from grid G used to discretize the space.

This simple rule is valid for closed surfaces, however, if the surface is known to have boundaries, the distance function must take this into account and return an *undefined* value for points outside the boundaries. In this case, assuming we have a ρ-dense point set (all balls having a radius ρ and its center in one of the sample points with at least another input point inside them) and that the measures are δ-noisy (errors from measures to original points are at most δ), a point p is marked as undefined if the projection of the point onto its closest tangent plane is greater than $\rho + \delta$. The implicit formulation is later triangulated using a surface meshing approach based on Marching Cubes.

The presented algorithm has two major problems. On the one hand, relying on the k-nearest neighbors to compute the tangent planes may fail in the case of having two



(a) Point set (b) Tangent planes (c) Hoppe's Reconstruction

Figure 4.11: Hoppe's method [104] applied to the Cat dataset. From left to right, the input point set, a visualization of the tangent planes (as small patches, the blue vector denoting the normal direction) and the reconstruction result.

parts of the sampled surface very close to one another, and of course, it does not assume outliers to be present in the data. On the other hand, the orientation of the normal is also a heuristic depending on k-nearest neighbors. Nevertheless, in case of close-to-ideal data, the method achieves smooth results (see Figure 4.11).

Natural Neighbors The other method in this section is the Natural Neighbors interpolation of Boissonnat and Cazals [29]. The Natural Neighbors of a point p_x are defined as the neighbors of p_x in the Delaunay triangulation of the set P. An equivalent definition is to describe the Natural Neighbors as the points on $P \oplus \{p_x\}$ whose Voronoi cells are modified upon insertion of p_x to the set. On the other hand, the Natural Regions $NR(V_{p_i}, p_x)$ of p_x are the portion of the Voronoi cells that are chopped off by this insertion. If $vol(NR(V_{p_i}, p_x))$ represents the volume (in 3D, area in 2D) of the $NR(V_{p_i}, p_x)$, then the Natural Coordinate associated with p_i is defined as follows:

$$\lambda_{p_i}(x) = \frac{vol(NR(V_{p_i}, p_x))}{\sum_{j \in P} vol(NR(V_{p_j}, p_x))}, \tag{4.12}$$

where consequently $vol((V_{p_j}, p_x)) = 0$ if p_j is not a natural neighbor of p_x.

The proposed Natural Neighbor interpolation defines an implicit formulation such that $\forall p \in P$, $I(p) = 0$, which is the following:

$$I(p_x) = \sum_i \lambda_{p_i}^{1+w}(p_x) f(p_i, p_x), \tag{4.13}$$

where w is some arbitrarily small value $w > 0$. In order to bound the natural coordinates dealing with unbounded Voronoi cells, 4 points on an enclosing bounding box around the input points are added to the point set. The choice for $f(p_i, p_x)$ is similar to that proposed by Hoppe et al. [104]:

$$f(p_i, p_x) = (p_i - p_x) \cdot n_i, \tag{4.14}$$

that is, the distance to the tangent plane defined by each oriented point. The local tangent planes are assumed to be known, since the method assumes known unit normals at input points.

Since the Delaunay triangulation of the points is needed to compute the Natural Coordinates of the points, they also use it to approximate the unknown surface. Taking its dual Voronoi diagram, the process evaluates the function at both endpoints of its edges, and retains those edges evaluating positive in an endpoint and negative on the other. The dual triangles in the Delaunay diagram of these Voronoi edges form the initial approximation $\tilde{S}$ of the surface.

This initial approximation $\tilde{S}$ can be refined by inserting points to the initial set and updating the Delaunay triangulation built in the last step. Note that the insertion of new points in the original Delaunay triangulation would alter the implicit function, so these new points are inserted in a new Delaunay triangulation. Each of the facets is associated with an error, which is the value $I(center(t_i))$, being $center(t_i)$ the center of the circumcircle of the facet. Facets whose errors are larger than a user's specified threshold, and facets whose angles are all smaller than 30 degrees are added to a queue. While this queue is not empty, the triangle t_i having largest error is extracted. Then, the intersection point between the Voronoi edge and the triangle t_i that is intersected in $\tilde{S}$ is added to P, and both the Delaunay triangulation and the queue are updated if needed.

To sum up, the reconstruction is performed by following a similar approach to that of Hoppe et al. [104], but by building the Delaunay triangulation of P and evaluating the implicit function at each of its tetrahedra circumcenters. The result is a set of triangles that can be further refined using the method of Chew [52], a precursor of the meshing algorithm presented in Section 3.4.2, to obtain the final mesh.

Markov Random Field Another interesting proposal revisiting the method of Hoppe el al. [104] is that of Paulsen et al. [169]. In their method a similar PCA pipeline is described, but slightly different heuristics are followed. Furthermore, the final distance function is regularized using a Markov Random Field (MRF) strategy, allowing the inclusion of the noise model of the capturing device into the process.

As in the original method, tangent planes approximating S are computed using PCA. However, the nearest neighbor search is restricted. First, a sampling density is computed by finding the average μ_i and standard deviation std_i of the distances to $K(p_i)$. Additionally, a fixed search radius of $2.5\mu_i$ is used to get the neighbors for a point. Furthermore, an outlier detection procedure is proposed. A point is considered an outlier if its third associated eigenvalue explains more than 10% of the variation, or its distance to the computed tangent plane is greater than μ_i. Then, a graph representation is built from the neighbors of a point. A point is considered as a neighbor of another if they are at a distance less than $\mu_i + 6std_i$ and their mutual normal angle is less than 15°. The connected components of the resulting graph containing fewer points than 1% of P are removed. Normal directions are then forced to be coherent with the position of the scanning device, so this information is assumed to be known.

Besides, not just the distance to the first nearest tangent plane is computed, but a blending to the 5 nearest ones is proposed. Depending on the nature of the data, two methodologies are proposed. The first one, amenable when the points are only influenced by Gaussian noise, consists of using the mean of these values. The second one, recommended for datasets containing outliers, is to use the median value.

At this point, the signed distance field is defined, but the level set representing the surface may not be smooth and contain holes. Thus, this initial implicit representation is refined through a MRF regularization. Given the initial implicit distance field I^0, the goal is to find the distance field I that maximizes the posterior probability defined by the theorem of Bayes:

$$I = \arg\max_{d} \mathcal{P}(I|I^0). \tag{4.15}$$

The local probabilities affecting it are formulated as energies. There are basically a prior and an observation model. On the one hand, the prior model defines the behaviour of the surface under no sampling. It is based on minimizing differences between neighboring Laplacians:

$$E_U(I(v_i)) = \sum_{GN_6(v_i)} \|L(v_i) - L(v_j)\|, \tag{4.16}$$

where $L(v_i)$ is the Laplacian of the function at voxel v_i, computed using finite differences. On the other hand, the observation model forces the surface to pass through P:

$$E_O(I(v_i)) = \left\|I(v_i) - I^0(v_i)\right\|, \tag{4.17}$$

that is, the difference between the original distance value $v_i^0 \in I^0$ and the current one. To balance the prior model, each voxel has an associated confidence value α_i computed using the distance to the $K^P(v_i)$, so that voxels near the input points get more confidence and vice versa.

Given the initial distance field I^0 and the prior and observation models, the Bayes' theorem provides inference on the posterior probability:

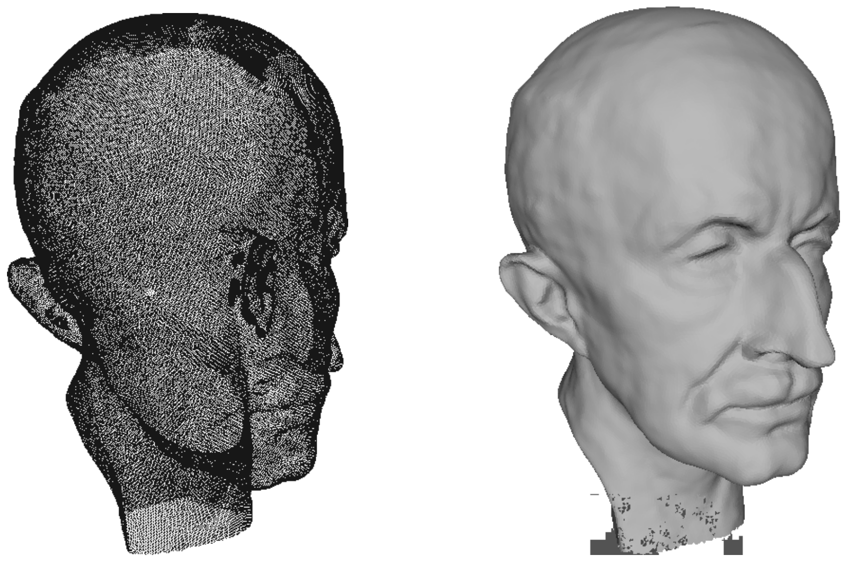

(a) Point set (b) MRF surface

Figure 4.12: Example of the MRF surface reconstruction method [169].

$$\mathcal{P}(I|I^0, GN_6(v_i)) = exp(-\alpha_i\beta E_O(I(v_i)) - (1-\alpha_i\beta)E_U(I(v_i))), \tag{4.18}$$

where β is a user-defined global weight. Note that by modelling the system as a MRF, the local neighborhood $GN_6(v_i)$ is sufficient to compute the maximum likelihood estimate of the previous probability for a given voxel. Furthermore, the maximization in Equation (4.15) can be solved by minimizing the weighted sum of the energy functions in Equations (4.16) and (4.17). The solution of this formulation is linear, and three methods have been proposed by the authors to solve it: Multiscale Iterative Conditional Modes, Conjugate Gradient and Sparse Cholesky Factorization. A discussion on these methodologies and the obtained results are stated in the original reference. Figure 4.12 shows a result obtained by this method.

Voronoi-Based Variational As pointed out in Amenta and Bern [10], under close to ideal sampling conditions, the elongated shape of the Voronoi cells spanned by the input points allows the approximation of their normals using the poles. However, this is not the case for noisy samples. In noise-ridden data, the Voronoi cells close to the data are not elongated anymore. However, Alliez et al. [8] observe that, by joining a set of these cells, its union finally resembles an elongated shape from which a normal estimation can be extracted.

In order to obtain a normal for each point, they start by computing the covariance matrix of its corresponding Voronoi cell. Then, the anisotropy of the covariance matrix is computed using the ratio between its largest/smallest eigenvalues. By iteratively adding new neighboring cells, the covariance matrix and its anisotropy are changed. The process continues until the union of cells having the largest anisotropy is the one retained, and a normal value is extracted from its covariance matrix [104].

Note that the gathered normal information is unoriented, and thus the set of retrieved covariance matrixes C defines a directional field. However, using this information only, the final shape inference is computed by maximizing the following constrained energy functional:

$$\begin{aligned} \arg\max_{f} \quad & E(f) = \int_{\Omega} \nabla f^T C \nabla f \\ \text{subject to} \quad & \int_{\Omega} \left[\|\lambda f\|^2 + \epsilon \|f\|^2 \right] = 1, \end{aligned} \tag{4.19}$$

where $E(f)$ is the anisotropic Dirichlet energy, accounting for the alignment between ∇f and the directions induced by C. Furthermore, the latter constraint imposes $E(f)$ to be maximized over the unit ball defined by the biharmonic energy. The expression is discretized and solved in an adaptive tetrahedral grid constructed using restricted Delaunay meshing in 3D, which is tuned to gracefully adapt the shape of its tetrahedra to the local sampling density (i.e., to follow an octree-like behaviour).

4.5.2 Unsigned Distance

Methods in this category rely on defining an unsigned distance function from the point sets, and then extract the surface from this representation. Note that using an unsigned distance field usually prevents the requirement of known per point oriented normals at input points. However, some heuristics have to be defined in order to extract a surface mesh out of this uncommon distance field.

Unsigned Graph Cut In the paper by Hornung and Kobbelt [107] the unsigned distance function is used as a confidence map representing the probability of the surface passing through a given voxel. From this probabilistic scalar field, a graph representation is computed and a S-T algorithm [37] is applied to divide the space into the *inside* and *outside* subspaces defined by the object.

The unsigned distance function is generated iteratively, by placing the points inside a voxelized space and then applying dilation operations over the 6 neighbors of a filled voxel. The diffusion of the values starts with voxels containing a point having $v_i = 0$, and then neighboring voxels are modified with the following function:

$$v_i = \frac{1}{7}\left(I(v_i) + \sum_{v_j \in GN_6(v_i)} I(v_j) \right). \tag{4.20}$$

The dilation operations are executed until a unique closed crust of voxels dividing the inside and outside space in the voxel grid is obtained. Then, a graph is built on this voxel space, where a node represents one of its faces, and each of them shares an edge with the neighboring faces in the same voxel. This generates an octahedral graph for each voxel, having an assigned weight of:

$$w(\mathsf{e}(\mathsf{v}_i, \mathsf{v}_j)) = I(v_i)^s + a, \tag{4.21}$$

where v_i is the voxel where the edge $\mathsf{e}(\mathsf{v}_i, \mathsf{v}_j)$ lies, the parameter s is used to emphazise the voxel values, and a is a constant. Then, an S-T algorithm [37] is applied to minimize the energy:

$$E(S) = \int_{\tilde{S}} v_i dv_i + \int_{\tilde{S}} a d\tilde{S}. \tag{4.22}$$

The surface voxels are defined as those containing at least one cut-edge after applying the Graph Cut algorithm. In order to speed up the process and to achieve hole filling and fine details, a coarse-to-fine hierarchical approach is proposed. Starting with a coarse voxelization of the space, the surface voxels of the lower resolution are used to initialize the crust of the higher resolution problem. This helps in the initialization of the proper crust and in filling gaps in low sampled areas.

The authors also propose a method to directly retrieve the mesh out of the surface voxels. In this mesh, vertices lay on voxel centers and faces correspond to voxel corners. The mesh is generated by running over all $2 \times 2 \times 2$ groups of voxels. Each center voxel is a polygonal face if the block contains at least 3 voxels from the set of surface voxels. The edges of the face are generated by cycling through the $2 \times 2 \times 2$ block of voxels.

Signing the Unsigned The method of Mullen et al. [159] proposes a completely different approach. Starting from an unsigned distance, it tries a set of heuristics in order to transform it into a signed distance field. The algorithm is divided into three parts. First, a rough unsigned distance function is computed from the points. Then a stochastic sign estimation, along with a confidence, is generated for each of the discrete distances. Finally, a globally coherent distance function is computed.

The initial rough approximation of the distance function uses the Wasserstein distance, defined as follows for an arbitrary point $p_x \in \mathbb{R}^3$ to the point set P:

$$dist_w(p_x) = \sqrt{\frac{1}{k} \sum_{p_i \in K(p_x)} \|p_x - p_i\|}. \tag{4.23}$$

This approximation, despite being robust to both outliers and noise, is not directly usable to extract the surface, since the distance is unsigned and has no precise isovalue defining $\tilde{S}$. However, it allows the identification of an *ϵ-band* (or volumetric crust) on the space defining where the densely sampled areas are located.

The working space is partitioned using an octree decomposition, to build a first coarse mesh D_1 using a Delaunay triangulation as the underlaying structure. Then, an optimal ϵ is sought using the following function:

$$m(\epsilon) = \frac{c(\epsilon) + h(\epsilon) + g(\epsilon)}{d(\epsilon)}, \tag{4.24}$$

where $c(\epsilon)$ is the number of components, $h(\epsilon)$ is the number of cavities, $g(\epsilon)$ is the number of tunnels and $d(\epsilon)$ is the density of the input points inside the ϵ-band. To compute an optimal value ϵ in $m(\epsilon)$, nodes of the coarse mesh are sorted according to their distance value and the range of possible ϵ is split into 200 intervals. The points, edges, faces and tetrahedra are bucket-sorted inside the corresponding intervals. Then, a union-find algorithm is used to compute the evolution of the connected components in each band. The density of the input points is computed as the ratio between the number of input points inside the ϵ-band and its volume. Finally, a union-find algorithm is used in reverse order of distance to get the number of connected components, from which the cavities $h(\epsilon)$ can be deduced. The values of $m(\epsilon)$ are plotted for the different values of ϵ, and the best ϵ is selected as the first local minima after the first local maximum.

Given an optimal ϵ-band, the distance inside it is refined in two ways. First, the mesh D_1 is extended using Delaunay refinement to contain twice as many vertices inside the ϵ-band, creating D_2. Second, the distance estimate inside the band is refined by using PCA. For each vertex in the band, the unsigned distance is recomputed as that to a plane created using nearest neighbors and PCA [104].

Once the consistent unsigned distance field I_u is estimated, the sign of the function needs to be retrieved for proper meshing. A first stochastic coarse estimation of the sign is performed. For each vertex in D_2 outside the band, a series of rays are thrown in different directions, and the number of intersections with the band is computed. If this number of intersections is odd, this means that the point is inside the surface. Otherwise, if it is even, this means the query point is outside. A confidence for this sign is also assigned by computing:

$$conf(p_i) = 2\max(e, o)/r - 1, \tag{4.25}$$

relating the number of even e and odd o intersections with r, the number of rays shot. To propagate this sign inside the band, all the vertices contained are sorted by their distance. Then, always taking the one with the largest distance, the neighbors having an already estimated sign $\hat{s}(p_i)$ are taken into account. If these neighbors have the same sign estimation, this estimation is assigned to the vertex along with a confidence equal to the greatest confidence of the neighbors. Otherwise, if some neighbors disagree on the sign, the confidence of the point is set to 0.

The final signed implicit function computation consists of a smoothing step over the previously computed signed function. This smoothing is performed by minimizing the following energy:

$$E(s(p)) = \int_{\Sigma} \left[|s(p)I_u(p)|^2 + W(conf(p))(s(p) - \hat{s}(p)) \right], \tag{4.26}$$

where $W(x)$ is a weighting function mapping the confidence $\hat{s}(p)$ to a weight on the fitting term, and $s(p)$ is the final sign. This minimization can be discretized in D_2 and solved using a system of linear equations.

The method of Mullen et al. [159] accepts outliers and constant noise, but fails when noise varies inside the dataset. Clearly, the fixed k-neighborhood used to compute the distance function does not take into account this variability, and should be adaptively devised according to the noise in a given region. In Giraudot et al. [90], the unsigned distance function is built with adaptive k values that depend on the local noise scale. Since the unknown surface is a 2-manifold, for a given evaluation point, they increase the k value until the apparent dimension matches that of a surface. They also propose some new approaches to sign the unsigned field.

4.5.3 Radial Basis Functions

The Radial Basis Function (RBF) approximation procedure is used in a broad range of application areas. An RBF is a function where its value depends on the distance from a given origin (the RBF center). In order to approximate a given function, a set of RBFs are merged together, using some weights. The result is the description of the function in a compact representation having the form:

$$f(p_x) = \sum_{i=1}^{n} w_i \phi(\|p_x - c_i\|), \tag{4.27}$$

where ϕ is the RBF, c_i is its center, and p_x is the point where the function is evaluated. Then, given a set of discrete samples of the function to approximate, the problem is reduced to finding a set of centers (i.e., a set of primitives) and their associated weights.

Blobby Before going into detail with more complex examples using RBF approximation, we present a first approach by Muraki [160] dealing with the problem using a different, though similar, formulation. The problem here is to reconstruct/compress a single range image. In this case, a set of spherically symmetric fields are used as primitives, and the scalar fields from n primitives are merged, or rather summed, to get a value for a given point:

$$I(v_i) = \sum_{i=1}^{n} b_i e^{-a_i f(p_i)}. \tag{4.28}$$

In Muraki [160], the primitive $\phi(p_i)$ is a spherical symmetric field having the form:

$$\phi(p_x) = (p_x - c_i)^2, \tag{4.29}$$

where c is the center of the primitive. As we have seen, this field value defined by f is decaying exponentially with respect to the distance from c, with modifying parameters a and b. With these principal equations, the optimization problem is defined as minimizing the distance between measured points and isosurface values for the m measured points with depth values z of the scan:

$$E_{value} = \sum_{i=1}^{m} |I(v_i) - z|^2, \tag{4.30}$$

with m referring to the number of voxels. Using only this equation, there is no clue about which part of the surface, the *outside* or the *inside*, is being fitted. Another constraint that accounts for the gradient of the voxel grid to match the one described by the normals of the input points is added to the optimization:

$$E_{normal} = \sum_{i=0}^{m} \left| n_i - \frac{-\nabla I(v_i)}{|-\nabla I(v_i)|} \right|^2, \tag{4.31}$$

where $\nabla V(v_i)$, the gradient at a given point, is computed using its neighbors in the regular grid. Finally, a term accounting for the influence of each primitive is added to the optimization, in this way preventing the movement of the shape into non-sampled areas:

$$E_{shrink} = \left(\sum_{i=1}^{n} a_i^{-\frac{3}{2}} |b_i| \right)^2. \tag{4.32}$$

With all the terms described above, the energy to minimize remains as follows:

$$E = \frac{1}{m}(E_{value} + \alpha E_{normal}) + \beta E_{shrink}, \tag{4.33}$$

where the α and β parameters are a tradeoff for the influence of the E_{normal} and E_{shrink} terms respectively.

Note that this approach does not directly fit the formulation presented in Equation (4.27), since the spherical fit depends on two parameters a_i and b_i, so the final optimization needs to find 5 parameters (the 3D center c_i, a_i and b_i) for each of the desired primitives used to approximate the range image. In this problem, the number of primitives is also an unknown, and a large number of primitives leads to a difficult problem to solve. For this reason, the author proposes making an initial fit using one primitive, and then dividing it into two to increase the goodness of the fitting. This division is applied recursively until the desired approximation is achieved. The problem of this procedure is choosing the order in which the primitives must be subdivided. In order to know which subdivision is to be performed in a given state, they apply the minimization to all the possibilities and get the one minimizing the energy, which, of course, is also a computationally-inefficient solution.

Fast RBF The method by Carr et al. [43] uses the most similar approach to the generic RBF approximation technique, defined in Equation (4.27), in order to interpolate a signed distance function I described by the input points with oriented normals. Input points from P are used as RBF centers, and their values in the implicit function to approximate must be zero. Having the RBF centers defined, their weights are the only unknowns. Additionally to P, off-surface points at a given distance following the normal of the point are added, both inside and outside the surface, in order to prevent the trivial solution of I being zero everywhere. For each p_i, its off-surface points are placed at a distance lower than $dist(K_1(p_i), p_i))$, to avoid interferences between different parts of the surface. The authors notice that off-surface points are not really needed for every point, so if the value/orientation of a given normal is not reliable, its corresponding off-surface points are not computed. The RBF system of equations related to this problem is the following:

$$\begin{pmatrix} A & Pol \\ Pol^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}, \tag{4.34}$$

where:

$$A_{i,j} = \phi(\|p_i - p_j\|) \tag{4.35}$$

$$Pol_{i,j} = pol(p_i), \tag{4.36}$$

and $pol(p_i)$ is a polynomial of small degree. Recommended RBFs for this problem are biharmonic splines, $\phi(r) = r$, where r is the radius, or triharmonic, $\phi(r) = r^3$. The use of RBF methods have been limited to datasets having a few thousand points, given the huge amount of required memory and computational cost encountered in solving the previously presented system of equations. To reduce this drawback, they propose using the Fast Multipole method [94] to solve the system of equations.

In order to reduce the computational cost of evaluating the function with a huge number of vertices, a center reduction procedure is also proposed. It consists of choosing only a subset of points as the RBF centers. Using the approximation they provide, residuals are evaluated, and new centers are appended in areas where they are higher. Then the RBFs are refitted and the insertion of new points is iterated until a user defined threshold is reached. Furthermore, the authors take into account the noise in the data by using a parameter ρ accounting for the stiffness to the given data points (i.e., smoothing). The system of equations accounting for parameter ρ is the following:

$$\begin{pmatrix} A - 8n\pi\rho\mathbf{I} & Pol \\ Pol^T & 0 \end{pmatrix} \begin{pmatrix} \lambda \\ c \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}, \tag{4.37}$$

where n is the number of centers, and $\mathbf{I}$ is the identity matrix.

Multilevel Partition of Unity One of the problems of applying approaches like the one presented in Carr et al. [43] is that the fitting of an RBF is global in the sense that all example input values are taken into account when finding the weights of a given basis function. This makes the resulting system of equations become dense, and computationally complex to solve. For this reason, there are some methods trying to alleviate this problem by using RBFs of compact support. However, Compactly Supported Radial Basis Function (CSRBF) are not able to close holes and fill in missing parts of the surface, which means that their influence domain has to be carefully selected.

There are several approaches having Ohtake as the first author dealing with the CSRBF problem in several ways. The common property to all of them is the addition of a fitting function associated to each of the RBFs. By doing this, local fitting is performed by a local function and the RBF values are used to blend the separated contributions. The most well-known method is Multilevel Partition of Unity (MPU) implicits [165]. In this algorithm, the space is subdivided using an octree, and functions are fitted to each local subdomain. Then, the influence of this function is weighted using an RBF of the form:

$$\phi(p_i) = b\left(\frac{3\|x - c_i\|}{2r_i}\right), \tag{4.38}$$

where $b(x)$ is the quadratic B-spline, and r_i is the support radius of this RBF. Since in this algorithm RBFs are only used as a weighting, the resulting approximation function depends on the RBFa as a multiplicative factor:

$$I(p_x) = \sum_{i=1}^{n} g_i(p_x)\phi_{i,r}(\|p_x - c_i\|). \tag{4.39}$$

The support of each weighting function is proportional to the diagonal of the cell of the octree where the approximation is computed ($r_i = \beta d$, β being a parameter and d

being the diagonal). For each cell during the subdivision process, a local shape function is built using least-squares fitting. The ball of radius r_i must contain a minimum set of points to compute the approximation. The radius is progressively increased by a factor until this minimum number of points is achieved. Then, three possibilities are proposed for the local approximation function:

1. 3D quadric: To approximate large, smooth areas of the surface.
2. Bivariate quadratic polynomial in local coordinates: To approximate a local smooth patch.
3. Piecewise quadratic surfaces: To fit edges and corners.

Tests on the distribution of the points and their normals are computed to determine which of the three functions to use in each case. If there are more than twice the minimum number of points in the cell, (1) or (2) are used, otherwise, more detailed checks based on the normals of the points have to be executed to decide if an edge or a corner is present. An example result of this method can be found in 4.13(b).

MPU+RBF Following the MPU approach, both methods presented in Ohtake et al. [166, 167] uses a Partition of Unity (PU) procedure followed by a finer fitting using an RBF procedure. The following equation, used in both methods, has a first term that is equivalent to the PU approach, while the second accounts for the RBF fitting:

$$I(p_x) = \sum_{c_i \in C} g_i(p_x)\phi_{i,r}(\|p_x - c_i\|) + \sum_{c_i \in C} w_i \phi_{i,r}(\|p_x - c_i\|), \tag{4.40}$$

where $g_i(p_x)$ is the local fitting function that has to be solved by least squares in a moving projection plane at c_i. The local projection plane is defined by a weighted averaging of normals, and the coefficients g we aim to solve in this local plane are the following:

$$h(u, v) = g_1 u^2 + 2g_2 uv + g_3 v^2 + g_4 u + g_5 v + g_6. \tag{4.41}$$

Then, the methods diverge in the procedure to obtain a reduced number of approximation centers C. In Ohtake et al. [166] they use a multi-scale approach and solve the problem in a coarse-to-fine way. At each scale, points are divided using an octree, and for each cell a single representative point is constructed using the centroid of the points falling on it and their mean normal. Then, starting from $I^0(p_x) = -1$, the interpolation functions at each level k in the hierarchy are:

$$I^k(p_x) = I^{k-1}(p_x) + o^k(p_x), \tag{4.42}$$

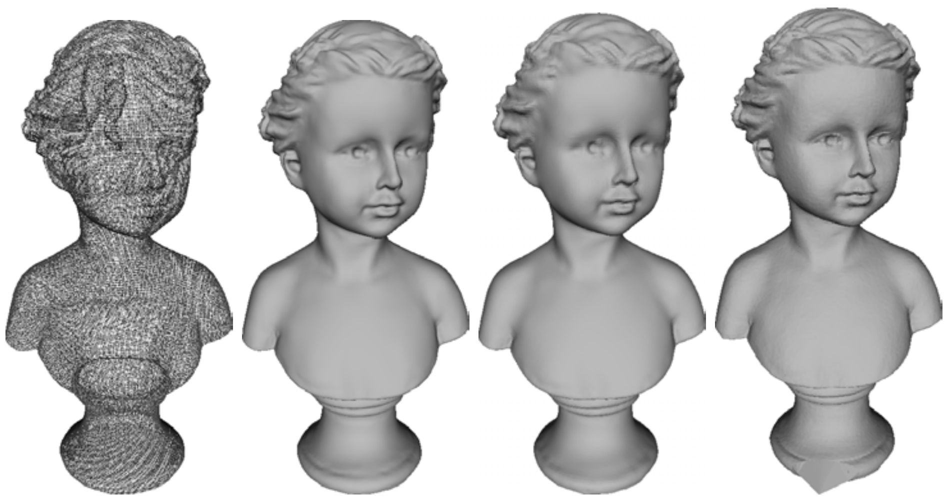

(a) Bust dataset (b) MPU (c) Adaptive (d) Multi-scale

Figure 4.13: Results of the reconstruction of the Bust dataset (a) for three similar methods: MPU [165] (b) and the adaptive [167] (c) and the multi-scale [166] (d) version of MPU+RBF.

where o^k is the octree node at depth k, having

$$o^k(p_x) = \sum_{c_i^k in C^k} [g_i^k(x) + w_i^k] \phi_{i,r}^k(\left\| p_x - c_i^k \right\|). \tag{4.43}$$

Coefficients w_i^k are found by solving the system of linear equations generated by $I^{k-1}(c_i^k) + o^k(c_i^k) = 0$. The support size r^k at each level is defined by $r^{k+1} = r^k/2$, where r^1 is proportional to the diagonal of the bounding box defined by P.

On the other hand, the method proposed in Ohtake et al. [167] uses a subset of the input points as C, as suggested by Carr et al. [43]. The selection of the centers is guided by a measure of the overlap at a given point p_x, corresponding to the evaluation $I(p_x)$ of the RBFs at this point. At each step, from a random subset of points having $I(p_i) > t$, the one with the lowest value is selected. Then, an optimal support size and local approximation are computed at that point. These values are used to update $I(p_i)$ for all $p_i \in P$, and the procedure is repeated until all $I(p_i) < t$. The selection of an optimal support size at each step is guided by the local error ϵ_{local} at each center. Thus, the energy to minimize is the following:

$$E(r) = \epsilon_{local}(r)^2 + \frac{d}{r^2}, \tag{4.44}$$

where d is proportional to the diagonal of the bounding box of P. As the support size increases, so does the local error, and the one dimensional minimization in $E(r)$ accounts for finding a good tradeoff between the two.

A simple test of both methods can be found in Figure 4.13 (c,d).

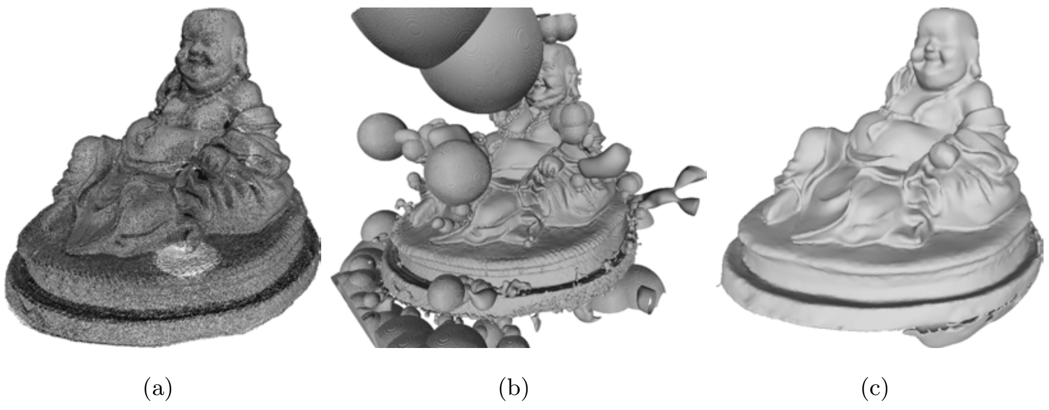


Figure 4.14: Comparison between the original MPU [165] and the smoothed version [161]. Note how the heavy noise of the input dataset (a), in both position and normal, leads to a bumpy surface with artificial components on MPU (b). Smoothing out the local contributions results in a continuous representation of the surface (c).

Smoothed PU The blending of local functions proposed in the MPU method [165] and its variants [166, 167] accounts for obtaining a continuous surface by merging these local contributions. However, these local approximations are computed independently, without taking into account that neighboring local surfaces should present a continuity of surface. The method by Nagai et al. [161] tries to overcome this phenomenon by smoothing the local contributions with respect to their vicinity.

Thus, local surfaces are smoothed out to better agree with one another in a two-step procedure. Only planar local approximations are used in order to simplify calculus, as the only parameters to modify are its origin point and normal. First, the normals are modified using Laplacian smoothing, i.e., each normal is replaced with an average of the normals of its neighbors. The new normals define a new gradient field, and the origin point is moved so that this new vector field corresponds with its gradient. These two steps are iterated (five iterations suggested) in order to obtain a smoothed version of the mesh.

Furthermore, the framework is reformulated into a covering spheres data structure, substituting the typical octree structure. Thus, all the differential operators needed are redefined to work in this new space division.

However, applying the previous method provides an over-smoothed mesh, which leads to loss of details and sharp features. To solve this problem, new terms are taken into account. First, they use an anisotropic weighting in order to recover the edges that might have been present on the original shape. Second, they introduce a data term into the smoothing in order to preserve small details. Note the improvement provided by this method over the original MPU in Figure 4.14.

Voronoi Centered RBF In the Carr et al. approach [43] the relevance of a good selection of centers for the RBFs is taken under consideration. Samozino et al. [181] propose a method that gracefully mixes this approximation method with the shape analogs for MA provided in some of the methods in Section 4.4.1. Instead of a subset of input points, the centers for the RBFs are selected from the poles in the 3D Voronoi diagram. This approach is based on the previous observation that the 3D MA can be approximated using the poles under certain conditions. They claim that using the RBF centers placed on the MA greatly reduces the number of RBFs required to approximate the shape faithfully.

Furthermore, instead of using the poles directly, a λ-medial axis is used, as defined by Chazal and Lieutier [49]. Naively speaking, the λ-medial axis is that having maximal empty balls of radius λ. This minimum on the empty ball size confers a more stable medial axis under small variations on the surface. Finally, given a desired number of RBFs, the poles in the set contained in the λ-medial axis are clustered to get the centers for the RBFs (compactly supported in this case). Since points on the MA contain the points with two or more closest points to the surface, the value for an RBF is imposed to be the radius of its corresponding medial ball.

Note that, while no per-point normals are required, a labeling of in/out poles similar to that of Power Crust [13] is needed to provide the system of equations with both negative/positive constraints.

Cone Carving RBF Due to the lack of data, even when using RBFs with full support on $\mathbb{R}^3$, the hole filling structures that arise in areas of poor sampling may not be correct. This is because no constraints are selected on these positions. On regular RBF interpolation, positive/negative constraints are located along the normal of each point. Consequently, when there are no points sampling a given part of the surface, there will be no constraints in the system. This causes the RBF procedure to create smooth surfaces over these areas, which may not be desirable depending on the context.

The method of cone carving from Shalom et al. [189] tries to ameliorate the results of the RBF interpolation method for undersampled parts by adding further constraints derived from visibility information. A visibility cone is the empty generalized cone with its apex at an input point p and formed by all the rays extending towards the outside of the shape. Since building the cone requires the unknown surface, an approximation of this surface is computed by splatting the points on an outer cube centered at a given point p. Then, the visibility cone is built by joining p with the silhouete of the parts of the cube not contained in the footprint of any splated point.

Taking into account a blended distance to the nearby visibility cones, new inside/outside constraints can be placed anywhere in the space to further improve the RBF approximation.

Eigen RBF Another approach toward removing the need of known normals in the RBF procedures is the method presented by Walder et al.[206]. The optimization is cast to the solution of an eigenvalue problem $Av = \lambda v$, resulting from an energy minimization taking into account many different terms. The resulting surface is well defined over the whole domain and the RBFs have compact support.

In order to avoid normals, the formulation they propose is the following:

$$\arg\min_I \quad \begin{array}{c} \underbrace{\sum_{p_i \in P} I(p_i)^2 -}_{\text{Data Term}} \\ \underbrace{\lambda_e \int_{u \in \mathbb{R}^3} I(u)^2 d\mu(u)) -}_{\text{Energy Term}} \\ \underbrace{\lambda_g \sum_{p_i \in P} |\nabla I(p_i)|^2 +}_{\text{Gradient Term}} \\ \underbrace{\lambda_r |I|^2}_{\text{Regularization Term}} \end{array} \tag{4.45}$$

Let's now define the purpose of each term in the function in Equation (4.45):

- **Data Term:** Values at input points should tend to zero.
- **Energy Term:** Values at points other than P should be relatively large.
- **Gradient Term:** Gradients of the resulting function applied at P should be large.
- **Regularization Term:** Restricts the smoothing by using thin plate energy [43].

Weights λ_e, λ_g and λ_r tune the strength of each term.

The function $I(x)$ has the form of a mixture of RBFs, as in Equation (4.27), but of compact support. The set of centers and supports in CSRBF (B-splines) are known. Contrary to other approaches, the RBF centers do not correspond to input points. Instead, they are distributed around the whole domain in an adaptive grid: RBFs with larger support and more spread are used in places far from the input data, and are smaller and more compactly supported in the inverse case. Although not specified, centers seem to be placed in a regular space subdivision structure, such as an octree for instance, their support being a multiple of their corresponding cell width or diagonal.

The variables to solve for are the weights w_i. Thus, the objective function has to be rewritten in terms of $w = [w_1, ..., w_n]$. By means of rewriting each term in a vector-matrix-vector form (see original text for details), the problem transforms to the following eigenvalue problem:

$$(-\lambda_e A_e - \lambda_g A_g + \lambda_r A_r + A_d A_d^T)w = \lambda w, \tag{4.46}$$

where A_e, A_g, A_r and A_d are the matrix forms of the data, energy, gradient and regularization term respectively, as a function of w. The solution is found in the smallest eigenvalue.

Despite the fact that this method does not require known normals, it does require the tuning of the λ parameters in Equation (4.45).

Support Vector Machines A different group of methods are those using RBFs as part of a machine learning technique, like the one by Schölkopf et al. [186], which has also been described in other papers such as Eigensatz et al. [74]. In this method, the surface reconstruction process is cast into a Support Vector Machines (SVM) learning problem. Its main advantage over other methods in this category is that it does not require normal values at input points. The idea is to use a positive definite kernel function allowing the transformation of the points into a Reproducing Kernel Hilbert Space (RKHS), with the hope that computations can become linear in this space. The positive definite kernel to use is the Gaussian one:

$$\phi(p_x, p_y) = e^{\left(-\frac{\|p_x - p_y\|^2}{2\sigma^2}\right)}. \tag{4.47}$$

The formulation of single-class SVMs is very similar in form to the main formulation of the RBFs presented in Equation (4.27):

$$I(p_x) = \sum_i \phi(\|p_x - p_i\|) - \rho. \tag{4.48}$$

However, the minimization problem to solve is completely different. The main idea comes from the one-class SVM, which tries to separate points from the origin by a hyperplane with the largest possible margin. This problem can be written using dot products, which is find the w and ρ such that $\forall p_x^H \in P^H, \langle w, p_x^H \rangle > \rho$, where P^H are the input points in a Hilbert space. A Hilbert space is a generalization of the Euclidean space to arbitrarily high dimensions. The requirements for a space to be considered a Hilbert space is to be a vector space allowing to compute inner products. Note that the distance to the hyperplane is $\rho / \|w\|$.

The problem of finding the SVM is extended in their proposal, and instead of only looking for a hyperplane, it aims to recover two: those enclosing all the data points. This approach is called the *Slab* SVM, and results in a convex quadratic optimization problem that targets finding the space between the two parallel hyperplanes containing P^H with a width $\delta / \|w\|$ guided by the user's parameter δ. Thus, the problem is to maximize the distance of the slab to the origin of the Hilbert space, which leads to the following minimization:

$$\begin{array}{ll} \arg\min_{w,p} & \frac{1}{2}\left\|w\right\|^2 - \rho \\ \text{s.t.} & \rho \leq \left\langle w, p_x^H \right\rangle \leq \rho + \delta \ \forall p_x^H \in P^H \end{array} . \tag{4.49}$$

This minimization is derived to its Lagrange dual optimization formulation (see original papers for the complete development):

$$\begin{array}{ll} \arg\min_{\alpha,\beta} & \frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \beta_i)(\alpha_j - \beta_j)\left\langle p_i^H, p_j^H \right\rangle + \delta \sum_{i=1}^{N} \beta_i \\ \text{subject to} & \forall i, \alpha, \beta \geq 0 \\ & \sum_{i=1}^{N}(\alpha_i - \beta_i) = 1. \end{array} \tag{4.50}$$

The whole the procedure described so far corresponds to a slab separation in a Hilbert space. Thus, for the moment, the problem uses a set of points mapped into an arbitrary Hilbert space. This means that the input points have been mapped from $\mathbb{R}^3$ to $\mathbb{H}$ by means of some mapping function, $map(p_x) = p_x^H$. Here is where the *kernel trick* can be used. It is known that certain functions, like Gaussians, can be expressed as an inner product. Thus, if the expression to solve is only defined according to dot products, the mapping from $\mathbb{R}^3$ to $\mathbb{H}$ does not need to be computed explicitly, and the dot product can be changed by the kernel function:

$$\langle p_i, p_j \rangle = \langle map(p_i), map(p_j) \rangle = \phi(p_i, p_j). \tag{4.51}$$

This substitution of the mapping for a kernel is the so called kernel trick.

As we have seen above, Equation (4.50) only depends on $\left\langle p_i^H, p_j^H \right\rangle$, so we can substitute it with $k(p_i, p_j)$. The implicit surface to solve then becomes:

$$I(p_x) = \langle w, map(p_x) \rangle - \rho = \sum_{i=1}^{N}(\alpha_i - \beta_i)\phi(p_x, p_i) - \rho, \tag{4.52}$$

where α_i and β_i are the solutions to Equation (4.50). Note that once α_i and β_i are known, the ρ variable can also be obtained.

One must notice that the presented Slab SVM method has some drawbacks. First, it involves solving a convex quadratic optimization problem. Second, the signed function is only well defined inside the slab, and tends to negative values in other places (either *inside* or *outside* the volume). And finally, the kernel function (i.e., the Gaussian) has global support.

Support Vector Regression Another approach using SVMs is the one presented in Steinke et al. [200]. In this case, normals at input points are required, since off-surface points, like those in Carr et al. [43], are used to constrain a Support Vector Regression

(SVR). The ϵ-insensitive SVM regression algorithm is used with this data in order to fit the surface (the reader is referred to Smola and Schölkopf [196] for a tutorial on SVR methods). The main property of the ϵ-SVR is that it penalizes the function values $f(x)$ if their difference with the expected value is above a threshold ϵ. The convex quadratic optimization corresponding to the ϵ-SVR formulation is the following:

$$\begin{array}{lll} \arg\min_{w\in\mathbb{H},\xi_i,\zeta_i\in\mathbb{R}} & \frac{1}{2}\|w\|^2 + c\sum_i(\xi_i+\zeta_i) & \\ \text{subject to} & \langle w, map(p_i)\rangle + \rho - I(p_i) & \leq \epsilon + \xi_i \\ & -\langle w, map(p_i)\rangle - \rho + I(p_i) & \leq \epsilon + \zeta_i \\ & \xi_i, \zeta_i & \geq 0 \end{array}, \tag{4.53}$$

where ξ_i and ζ_i are slack variables, and c is a parameter accounting for the tradeoff between error penalization and function regularization.

The authors propose using a slight variation of the standard ϵ-SVR, in which the displacement ρ is set to a fixed value in order to simplify the dual formulation of the optimization problem. The Lagrange dual to this problem, again using the kernel trick, is the following:

$$\begin{array}{ll} \arg\min_{\alpha_i,\beta_i\in\mathbb{R}} & \frac{1}{2}a^T Ma + \sum_{i=1}^{|P|}(\alpha-\beta)I(p_i) + \\ & +\epsilon\sum_{i=1}^{|P|}(\alpha_i+\beta_i) \\ \text{subject to} & 0 \leq \alpha_i, \beta_i \leq c, \end{array} \tag{4.54}$$

where $a = (\alpha_1 - \beta_1, ..., \alpha_n - \beta_n)^T$, $M(i,j) = \phi(p_i, p_j)$, and c is a user's parameter.

As in Scholkopf et al. [186], the hyperplane ends up being defined by the kernel functions (see Equation 4.52). The problem proposed in Equation (4.54) is solved by using a coordinate descendant optimization algorithm. The function is optimized one dimension at a time, keeping the rest of the values for the other variables fixed:

$$\alpha_i^{new} = \frac{-I(p_i) - \epsilon - \sum_{j\neq i} M_{i,j}\alpha_j}{M_{i,i}}. \tag{4.55}$$

This method is similar in essence to the Gauss-Seidel method. In order to ensure sparsity of the matrix M and providing less computational effort, the kernel function to use is a CSRBF. More precisely, the kernel is the Wu function, which has the form:

$$\phi(w) = (1-w)_+^4(4 + 16w + 12w^2 + 3w^3), \tag{4.56}$$

being $w = \frac{\|p_i - p_j\|}{r}$, and r the support size. In order to further reduce computational cost, the bounding box of the points is subdivided to a given resolution proportional to the support size, and initial centers are selected. Furthermore, the procedure is multi-scale: first a coarse approximation is sought, and then only its residuals are approximated on finer scales (the support of the kernels at different scales is halved).

4.5.4 Moving Least Squares

The Moving Least Squares (MLS) is a technique widely used in many fields requiring the approximation or interpolation of scattered points. Broadly speaking, the procedure consists of applying a Least Squares in a local reference frame at each point. Then these local fits are joined together by weighting their contribution by a function dependant on the distance from each generating point, that is, an RBF. The final approximation function $I(p_x)$ is obtained from a set of local approximation $I_i(x) = g_i^T(x)c(x)$, where $g_i(x) = [b_1(x), b_2(x), ..., b_n(x)]$ is the polynomial basis, and $c_i(x) = [u_1, u_2, ..., u_n]$ is the vector of unknown coefficients. Thus, $I(p_x)$ has to minimize the following equation:

$$I(p_x) = \arg\min_{I_i} \sum_{i=1}^{|P|} \phi(\|p_x - p_i\|) \, \|g_i(p_x)c_i(p_x) - I(p_i)\| \, . \tag{4.57}$$

Despite having this common definition, the term MLS surfaces has been used to define two different approaches. First, we find the methods using MLS surfaces as a projection operator. Given a point p_x, and by means of a non-linear optimization procedure, a projection of this point onto the MLS surface is found without computing the full minimization proposed in Equation (4.57). On the other hand, we find methods that try to solve the minimization proposed in Equation (4.57) directly, which leads to an implicit function $I(p_x)$ that can be evaluated over the whole domain. This means that, in this second approach, the function can be evaluated at discrete positions in space (e.g., a voxel grid), and extract the surface using a surface mesher. In order to distinguish between these two approaches, we refer to the first one as the projection MLS approach, and the second one as the implicit MLS approach.

As we can see, these two approaches, while similar in nature, differ in the final use of the procedure. As previously mentioned, the implicit MLS approach can be used directly for surface reconstruction purposes, and consequently they fit as part of the present review. On the other hand, the projection MLS defines a local method to get a point projected on an unknown surface defined by the samples. Obviously, this procedure is not directly applicable to the surface reconstruction problem as described in this survey. In our case of study, projection MLS methods are useful as a preprocessing of the points before a surface reconstruction method is applied, since they can be used for tasks such as upsampling, downsampling or noise reduction. However, these kinds of methods are widely used in current point-based visualization, that is, visualization techniques using points as primitives. This is because projection MLS allows the resampling of the surface at a desired sampling rate, and thus renders an image of the surface at a desired resolution.

For the above mentioned reasons, in the following, we focus on specific examples of MLS having an implicit formulation. Also, it is worth noting that, since all the MLS definitions are defined just near the points, bounded surfaces can be recovered at the limit of their support.

Point Set Surfaces Based on the idea of stationary points defined by Levin [137], the authors introduced the projection operator in the computer graphics literature in [5], with the name of Point Set Surfaces (PSS). Given a query point p_x near an unknown surface, its projection onto the approximated MLS surface can be computed using only a set of k-nearest neighbors from P. The projection operation is performed in two steps: first a local reference frame is computed, then a bivariate polynomial approximation is computed in this local frame, and finally the point is projected onto it.

In order to find the reference frame (or plane) H for a given $p_q \in \mathbb{R}^3$ (q for *query*) near S, the idea is to minimize the local weighted sum of squared distance of points p_i to the $H(x)$, that is, minimize the following function:

$$E(p_q, n) = \sum_{i=1}^{|P|} \phi(\|p_i - p_q\|)(n^T p_i - n^T p_q)^2, \tag{4.58}$$

where ϕ is the weighting function. The unknown p_q is formulated in terms of the query point p_x as a displacement s from it along n: $q = p_x + s \cdot n$. This leads to the following function to minimize:

$$E_{mls}(p_q, n) = \sum_{i=1}^{|P|} \phi(\|p_i - p_q - s \cdot n\|) \langle n, p_i - x - s \cdot n \rangle^2 . \tag{4.59}$$

Note that the weighting function $\phi(p_x)$ is a Gaussian having the following form:

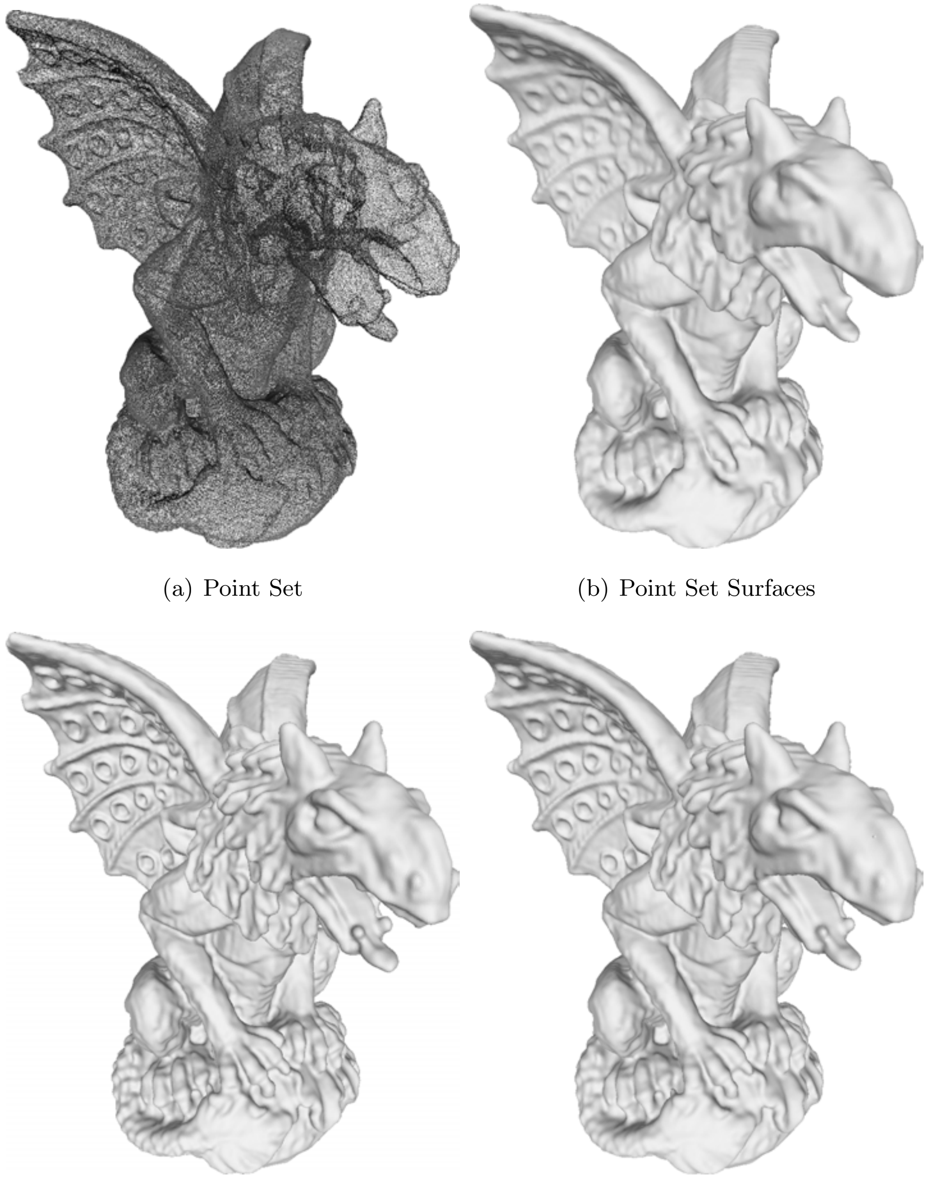
$$\phi(p_x) = e^{\frac{-p_x^2}{h^2}}, \tag{4.60}$$

being h a parameter reflecting the anticipated spacing between neighboring points. This parameter h serves as a tradeoff to smooth out features of the surface. Using non-linear iterative minimization, and prioritizing the smallest s, the required reference frame is retrieved.

On the other hand, to find the local bivariate polynomial fit $g(x, y) = z$, a standard least squares technique is applied. Each point p_i defines a height value $f(p_i)$ over the reference frame H, and the error to minimize is the following:

$$E_{mls} - h(g(x, y)) = \sum_{i=1}^{|P|} \phi(\|p_i - p_q\|)(g(x_i^H, y_i^H) - f(p_i))^2, \tag{4.61}$$

where, after the previous local frame computation, p_q is the projection of p_i in H, and (x_i^H, y_i^H) its local coordinates in H. Finally, the projection of the point is $p_x + (s + g(0, 0))n$.



(a) Point Set

(b) Point Set Surfaces

(c) Implicit PSS

(d) Algebraic PSS

Figure 4.15: Surface reconstruction example for some methods in the Moving Least Squares category. We used the original APSS [97] implementation (d), which also contains the original Point Set Surfaces [6] (b) and the Implicit PSS [124] (c). The implicit formulations posed by each variant are contoured through Marching Cubes [145].

The initial definition of the MLS projection operator by Alexa et al. [7] can also be cast to an implicit formulation. This procedure has been reinterpreted for simplicity by the same authors in various articles, by only using the reformulated first part of the algorithm, that is, the projection to a local reference plane. Based on this simplification, they propose an implicit formulation of their MLS definition [6] as:

$$I(p_x) = n_x(a(p_x) - p_x), \tag{4.62}$$

where $a(x)$ is the weighted average of neighbor samples:

$$a(p_x) = \frac{\sum_i^{KP(p_x)} \phi(\|p_x - p_i\|)p_i}{\sum_i^{KP(p_x)} \phi(\|p_x - p_i\|)}. \tag{4.63}$$

The normal direction n_x at p_x is computed as a weighted average of input normals, in the case of oriented point sets, or as the direction of smallest weighted covariance at p_x, for an unoriented set. See the smooth results provided by this technique in Figure 4.15 (b). Furthermore, they allow for boundaries by defining the surface just around a given distance from the point set [2].

Another interesting approach is the one by Amenta and Kil [14], casting the previously defined surface as the set of critical points of an energy function along lines defined by a vector field. Furthermore, they also claim that all the existing variants of MLS surfaces can be defined using an energy function paired with a vector field.

Finally, it is worth noting that the locality of this method has been used to develop out-of-core methods [59], working on slabs of the data fitting in memory to process large point sets.

Implicit MLS This variant of MLS was first defined by Shen et al. [191] to interpolate/approximate polygon soups. The system of equations derived from Equation (4.57) using the point set constraints, which are the ones we are interested in, is the following:

$$\begin{pmatrix} \phi(\|p_x - p_1\|) & & \\ & \ddots & \\ & & \phi(\|p_x - p_n\|) \end{pmatrix} \begin{pmatrix} g^T(p_1) \\ \vdots \\ g^T(n) \end{pmatrix} c = \begin{pmatrix} \phi(\|p_x - p_1\|) & & \\ & \ddots & \\ & & \phi(\|p_x - p_n\|) \end{pmatrix} \begin{pmatrix} I(p_1) \\ \vdots \\ I(p_n) \end{pmatrix}. \tag{4.64}$$

They solve for c using normal equations. Notice that this approach is similar to the MPU approach proposed in [165], and described in Section 4.5.3 of this chapter. In this case, a local fit is computed for each point, while in MPU the space is first partitioned using an octree and the fit is computed on each of its leaves.

Following the proposal of Shen et al., but applied to point clouds directly, we find the simple implicit MLS approach proposed in [124]. By taking $g = [1]$ and assuming

$I(p_i) = 0$, the definition they propose for the MLS implicit is the following:

$$I(p_x) = \frac{\sum_{i=1}^{|P|} \phi(\|p_x - p_i\|)(n_i^T(p_x - p_i))^2}{\sum_{i=1}^{|P|} \phi(\|p_x - p_i\|)}. \tag{4.65}$$

That is, each point defines a local tangent plane (given its normal), and the weighted distance from a point p_x to all the tangent planes defined by the points P is the value of the implicit MLS surface for p_x. The results of applying this simple procedure are depicted in Figure 4.15 (c).

By assuming a uniform sampling, the author demonstrates that the zero-set of the resulting implicit is isotopic to the original surface. Extending this work, Dey and Sun [68] propose using a weighting function (a Gaussian) dependant on the $lfs(P)$. By doing this, they can also obtain theoretical guarantees in the case of non-uniform sampling.

Algebraic PSS Another popular approach is the one presented by Guennebaud and Gross [97], where the APSS are described. As opposed to the previously presented proposals for Implicit MLS, they define the implicit function for each query point p_x as an algebraic spherical fit at this point. By fitting algebraic spheres instead of planes, the APSS achieves a non-iterative procedure to get an implicit surface representation as well as a projection operation. An algebraic sphere is defined by $s(p_x) = [1, p_x^T, p_x^T p_x]u$, where $u = [u_0, u_1, u_2, u_3, u_4]$ is the vector of coefficients.

The fitting procedure described uses normals to further constrain the solution. In case of unknown normals, an initial simple algebraic sphere is fit to a point using its neighbors. This initial fit is used to get an approximated unoriented normal as the gradient of the scalar field it describes. To get a coherent orientation along the point set, a $MST(P)$ is constructed, so the procedure is similar to the method by Hoppe et al. [104]. First, a confidence value is assigned for each of the normals, based on the residuals of their eigenvalues compared to their smallest positive one. Each edge on the graph has as its weight value the sum of the confidences of its two endpoints. Moreover, some of the edges in the MST are pruned, by removing edges from a point to a neighbor that is *behind* another neighbor, i.e., p_j is removed from the k-nearest neighbors of p_i if there is a point p_n so that $(p_i - p_n)^T(p_j - p_h) < 0$. This definition allows taking into account only the closest points to p_i in each direction. The propagation of the normals starts orienting outwards a point on the bounding box of the set, and traversing the tree. At each edge, the propagation of the orientations between points is performed by fitting an algebraic sphere as an approximation of the surface passing through the two points.

Once the normals are known, the authors propose including the constraint of having the values of the fitted sphere gradient equal to the normal values at each $p_i \in P$. This formulation results in a linear system of equations to solve:

$$W^{\frac{1}{2}}(x)Du = W^{\frac{1}{2}}(x)b, \tag{4.66}$$

where:

$$W(x) = \begin{bmatrix} w(p_x,p_1) & & & & & & & & \\ & \ddots & & & & & & & \\ & & w(p_x,p_n) & & & & & & \\ & & & \beta w(p_x,p_1) & & & & & \\ & & & & \ddots & & & & \\ & & & & & \beta w(p_x,p_1) & & & \\ & & & & & & \ddots & & \\ & & & & & & & \beta w(p_x,p_n) & \\ & & & & & & & & \ddots \\ & & & & & & & & & \beta w(p_x,p_n) \end{bmatrix} \quad D = \begin{bmatrix} 1 & p_1^T & p_1^T p_1 \\ \vdots & \vdots & \vdots \\ 1 & p_n^T & p_n^T p_n \\ 0 & a_1^T & 2a_1^T p_0 \\ \vdots & \vdots & \vdots \\ 0 & a_1^T & 2a_1^T p_n \\ \vdots & \vdots & \vdots \\ 0 & a_d^T & 2a_d^T p_0 \\ \vdots & \vdots & \vdots \\ 0 & a_d^T & 2a_d^T p_n \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a_1^T n_1 \\ \vdots \\ a_1^T n_n \\ \vdots \\ a_d^T n_1 \\ \vdots \\ a_d^T n_n \end{bmatrix}, \tag{4.67}$$

where a_x denotes the unit basis vector in dimension x of the coordinate system of the fitting reference frame, and β is a weight for the normal constraints.

It is worth noting that they also rely on a weighting function that has the following form:

$$w(p_x, p_y) = \phi\left(\frac{\|p_x - p_y\|}{h_i(p_x)}\right), \tag{4.68}$$

where the spacing $h_i(p_x)$ can be dependant on the sampling of P, and ϕ is recommended to be $\phi(r) = (1 - r^2)^4$, if $r < 1$ or zero otherwise. A sample result of this method can be found in Figure 4.15 (d).

In a later work from the same authors [96], they claim that the constraint of aligning the gradient and the normals of the points is sufficient to achieve the u_1, u_2, u_3 and u_4 from u, so only the u_1 component remains unknown. Thus, they first compute these 4 components, and then they use the algebraic distance constraints to find u_1. This 2-step process provides a faster fitting. Moreover, they propose multiplying the u_4 component with a user's parameter, which provides control of the curvature of the fitting.

Hermite PSS The previously introduced implicit definition of PSS is modified to allow for the interpolation of both point and normal constraints of Alexa and Adamson [4]. This is achieved by using singular weight functions, i.e., RBFs tending to infinity when close to zero. An example of such a function can be:

$$\phi(p_x, p_y) = \|p_x - p_y\|^{-m}, \tag{4.69}$$

where $m \geq 2$.

Using these weighting functions, they enforce the implicit function to take the previously required interpolatory constraints for each $p_i \in P$:

$$a(p_i) = p_i, \quad n(p_i) = n_i, \tag{4.70}$$

being $a(x)$ the centroid, as presented in Equation (4.62), $n(x)$ a function giving a normal at a point. That is, when evaluated at input points, the points themselves and their normals are preserved.

4.5.5 Deformable Surfaces

Deformable surfaces methods start the process with a dummy initial surface, usually a very rough approximation of the factual or just a simple shape, e.g., an sphere, enclosing or enclosed into P. Then, this initial shape is iteratively deformed towards the input point cloud.

The majority of these methods use an implicit form to represent the moving surface, since it allows making computations of a surface inside the cartesian grid without the need to parameterize the moving surface. Nevertheless, and despite the drawbacks commented in Section 3.2.1.2, there are also some methods using a deformable explicit surface.

Level Set A representative example of this section is the level set approach. The level set is a method for deformation of closed curves or surfaces in d dimensions. The tracking of the surface is done through its zero level set (i.e., the 0-isosurface) in its implicit representation. This implicit surface is dynamically deformed and tracked according to a given Partial Differential Equation (PDE). Thus, the level set method proposes a non-parametric variational optimization tracking a surface deformation on a regular grid.

Zhao et al. [217] were the first in using the level sets formulation for surface reconstruction. Later, the authors updated the method by speeding-up the techniques for the computation of both the level set and the initial surface [216].

Intuitively, during the level set method, the surface is iteratively deformed so as to minimize a given energy functional. Two independent models defining this energy are proposed by the authors. The first one, referred to as the minimal surface or the gradient flow model, is a variational formulation where the energy to minimize depends on the distance to the input data P and also a regularization term depending on the current isosurface $\tilde{S}$:

$$E(\tilde{S}) = \left[\int_{\tilde{S}} dist_d(p_x, P) area(S)\right]^{1/d}. \tag{4.71}$$

Using this energy term, the resulting surface behaves like an elastic membrane attached to P. The Euler-Lagrange formulation derived from the energy term in Equation (4.71) is used to derive the stationary condition related to the energy term:

$$dist_{d-1}(p_x)\left[\nabla dist_d(p_x)\cdot n_x + \frac{1}{d}dist_d(p_x)k_(p_x)\right] = 0, \tag{4.72}$$

where we can see a balance between the potential force, the first term in square brackets, and the surface tension, which corresponds to the second term in square brackets. Furthermore, the scaling factor $dist_{d-1}(p_x)$ provides the *membrane* to be more rigid away from the input data and more flexible when close to it. The parameter d, i.e, the norm used by the distance function, affects this flexibility.

The second model presented is the convection one. The presented gradient flow model in Equation (4.72) requires the computation of the mean curvature of the surface, which is a nonlinear parabolic equation. Following Courant-Friedrichs-Lewy (CFL) conditions [57], the time step for evolving the surface Δt is restricted to be less than h^2, where h is the resolution of the grid discretizing the working space. This results in lots of iterations being needed in order to reach the local minimum defining the surface. For this reason, the simpler and coarser convection model is recommended to be executed first, followed by the more complex and finer gradient flow model computation. The convection model of a flexible surface $\tilde{S}$ on a velocity field, which in our case corresponds to the distance function to P, is described by the differential equation:

$$\frac{dS(t)}{dt} = -\nabla dist(p_x, P), \tag{4.73}$$

these equations reflect that a point p_x, when attracted by this distance field, is attracted to its closest point in P.

The previously reviewed models define the energy functionals to minimize inside the level set method. Starting from a rough initialization of the surface, it is deformed towards the minimum following a motion model, i.e., it is iteratively deformed at different time steps. Thus, in order to apply the level set formulation we need to extract its time evolution (motion) as a PDE. In order to do so, the zero level set must have the same motion law as the moving surface:

$$\frac{dI(S(t),t)}{dt} = I(t) + \frac{dS(t)}{dt}\cdot\nabla I = 0, \tag{4.74}$$

where $\frac{dS(t)}{dt}$ represents the velocity of p_x in:

$$\tilde{S} = \{p_x : I(x,t) = 0\}. \tag{4.75}$$

The motion of the gradient flow and convection models has to be extended to all possible level sets of $I(x,t)$. Since in both models the motion depends only on the geometry of the moving surface, the motion is geometric, and the same one has to be applied to all level sets. Thus, the gradient flow level set formulation remains as follows:

$$\frac{\delta I}{\delta t} = |\nabla I| \left[\nabla d \frac{\nabla I}{|\nabla I|} + d \nabla \frac{\nabla I}{|\nabla I|} \right], \tag{4.76}$$

while the level sets formulation of the convection model is the following:

$$\frac{\delta I}{\delta t} = \nabla dist(p_x, S) \cdot \nabla I. \tag{4.77}$$

With the PDEs derived, an initial signed distance function is iteratively distorted using Equation (4.77) in the initial time steps, and later refined following Equation (4.76) in the final steps. Therefore, is important to get a reliable initialization of this signed distance function.

Starting with the unsigned distance defined from P, an outer isovalue far enough from the points to capture the shape of the surface is selected. Obviously, a closed shape defines two iso-contours in its unsigned distance field: one on the inside of the shape and one outside. The idea is to use the outer iso-contour to define the initial signed distance function. Using the knowledge of the bounding box defined by the grid I to be on the outside of the surface, the outer volume is propagated to the previously defined outer isovalue. Then, all the points not tagged as exterior are defined as interior, and a signed distance function to the isovalue can be built.

While the level set definition remains the same through the two proposals [170, 215], the initialization procedures differ. In their first approach [217], the unsigned distance field is built using the eikonal equation, upwind updates and Gauss-Seidel iterations using a different ordering of sweeps accross the discrete grid I. On their second approach [216], a fast tagging algorithm is proposed.

Note, however, that deforming the initial signed distance inside a discrete grid following either Equation (4.77) or (4.76) may produce the resulting implicit not being exactly a signed distance function. For this reason, a reinitialization procedure is applied after a number of iterations in order to maintain the evolving isosurface as a signed distance function.

Coulomb Potentials The method proposed by Jalba and Roerdink [111] uses a modified version of the fast tagging approach presented above, called the convection algorithm. Basically, it provides a scalable and memory efficient process by redefining the process to be performed in an octree structure. In this method, the term $dist(p_x, P)$ in Equation (4.73) is substituted by the electric scalar potential (a.k.a. Coulomb potential).

Each point in the input set has an associate charge value $q(p_i)$. Authors propose not using the Coulomb potential directly, but the generalized Coulomb potentials, which are able to decay faster with distance than as the inverse of distance:

$$I(p_i) = \sum_{j \neq i} \frac{q_j}{\|p_i - p_j\|^m}, \tag{4.78}$$

where $m > 1$ represents the order of the Coulomb potential. Higher order potentials are a good choice for ideal data, while lower order potentials are required in order to filter noise. Since the working space is subdivided using an octree, potentials are only evaluated at the centers of the nodes of the octree, instead of the full space.

Given a user's defined maximum octree depth, one point is inserted into the octree at a time. As previously mentioned, each point consists of a position p_x and a charge $q(p_x)$. Upon insertion, if the point reaches the maximum allowed depth for the octree, and that node is not empty, its centroid and total charge is assigned by taking into account all points/charges falling into it:

$$\begin{array}{ccc} centroid(o_x^d) & = & \frac{\sum_{p_x \in o_x^d} p_x q(p_x)}{\sum_{p_x \in o_x^d} q(p_x)} \\ q(o_x^d) & = & \sum_{p_x \in o_x^d} q(p_x) \end{array}, \tag{4.79}$$

where $centroid(o_x^d)$ and $q(o_x^d)$ are the centroid and charge of a given octree cell of depth d. Initially, all particles have the same charge $q(p_x) = 1$. Once constructed, the octree is balanced (i.e., any two neighboring cells differ at most by 1 in depth). In order to evaluate the Coulomb potential at any arbitrary position, the total charge and centroid of each of the cells of the octree must be computed. The octree is traversed in depth-first order and the Equations (4.79) are applied at each depth. Then, the evaluation is performed following the approach proposed in Barnes and Hut[18], which is briefly discussed in the following. To evaluate the potential at a given position p_x, a small positive test charge $q(p_x) = 1$ is assumed, and the following equation is applied:

$$I(p_x) = q(p_x) \sum_{i=1}^{|P|} \frac{p_i.q}{|r_i - p_i|^m}, \tag{4.80}$$

where r_i is the distance from p_x to the centroid of the ith cell. If the ratio $l/r_i < \theta$, l is the size of the current cell in the octree and θ is a user-defined threshold trading for accuracy, the potential is computed from the total charge and centroid of the particles falling within the cell. Otherwise, the computation continues with the children nodes of the current cell.

Once a way to obtain the potential at an arbitrary position has been defined, the convection algorithm can be applied. Of course, the algorithm must be redefined to work in the octree structure. The traversal is performed in a breadth-first manner, using a heap sorted by increasing value of the potential evaluated at each $centroid(o_x^d)$. Starting with a set of cells known to be at the exterior of the shape, the queue is built using the set of exterior cells that still have at least one untagged cell. Then, the front advances towards non-empty leaves following the values of I in increasing order, labeling the cells as exterior at each marching step. When the ridges and local maxima of I are reached, the marching stops and the current cells are labeled as part of the boundary. The remaining cells are labeled as interior.

Note that instead of initializing an outer surface and deforming it through a level set [216], the tagging algorithm is used directly to find the exterior volume of the surface isovalue defined by the previous potential. Thus, the interface in the volume corresponding to the surface is fit directly. Basically, the tagged volume is considered as the indicator function χ, which is smoothed out in order to alleviate the *stairwise* shape that would appear if the isosurface was directly extracted from it. The smoothed version $\widetilde{\chi}$ of χ is computed as the sum of contributions of nearby cells, where each contribution is weighted by quadratic B-spline kernels.

Touch-Expand Graph Cuts In latter years, there has been a greater popularity of S-T graph cuts methods to solve some variational formulations. In the present case, the method of Lempitsky and Boykov [135] propose an extension of existing S-T cuts approaches to work on a slab of the volume at a time, in this way reducing the memory requirements that would arise if using a whole discretized grid on the working space.

In this case, the energy to minimize is expressed as follows:

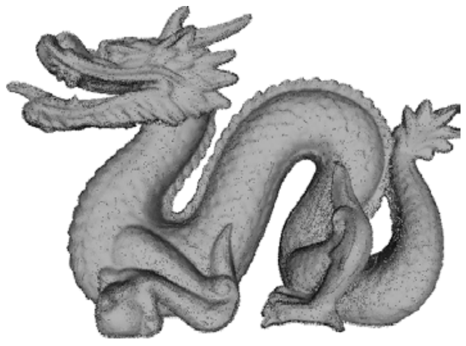
$$E(S) = \int_S \lambda ds - \int_S \langle \vec{v_s}, n_s \rangle ds. \tag{4.81}$$

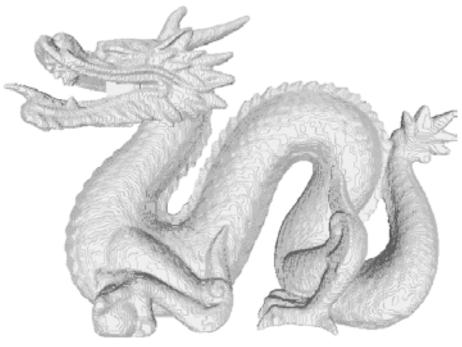
Below we discuss the meaning of this expression. The first integral of the energy is the shape prior, corresponding to the typical area-based regularization governed by the user's parameter λ. The second integral accounts for the data fitting and tries to maximize the *flux* functional. The flux is a measure of how well the surface is aligned with a vector field. In the present case, one can generate a semi-dense vector field generated from P and its associated normals. Here semi-dense means that each point and normal is blurred in the space using a Gaussian, to spread its influence. In Equation (4.81) this similarity between the normals over the surface n_s and the input vectors $\vec{v_s}$ is enforced through the dot product. Using a divergence theorem, the problem equals the maximization of the integral over the volume M bounded by S (i.e., the interior of S):

$$E(S) = \int_S \lambda ds - \int_M \nabla v_p dp. \tag{4.82}$$

An occupancy metric, derived from the lines-of-sight from the sensor or the silhouette information, can be further added to the minimization if available.

In order to minimize the previous energy, the touch-expand algorithm uses a max flow algorithm within a band around the points. The exterior/interior slabs needed to


(a) Point set


(b) Touch-expand result

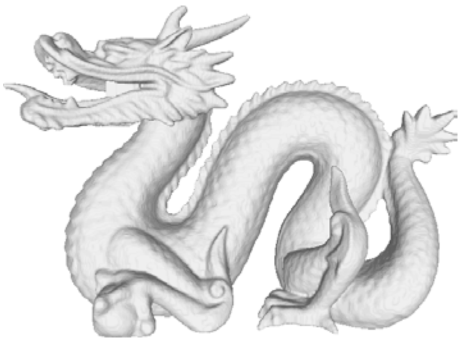

(c) Smoothed

Figure 4.16: Touch-expand algorithm [135] applied to the Stanford Dragon dataset (figure in the first row). Note how the resulting surface presents a staircase effect, produced by the method being a binarization inside a regular voxel grid (lower left figure). As suggested in the original reference, some smoothing is needed to alleviate this aberration (lower right figure).

connect the source/sink links (correspondingly separating the in/out sets) are detected using normal information, or even cruder notions like the scanning pose/direction. Then, the typical min-cut approach is computed to obtain a separation on the slab minimizing the given energy. If the initial band is too narrow in some parts of the object, the solution retrieved by the S-T cut *touches* one of the source/sink sets, meaning that the solution was not contained in the initial slab. Consequently, the slab is enlarged around the touched area, and the min-cut is computed again. This process continues iteratively until no *touches* are produced in the result, i.e., the optimal solution is within the current slab.

As opposed to other methods using graph cuts, like that of Hornung and Kobbelt [107], where the optimization is performed on a fixed band, this method ensures finding a global solution, as it extends the band on demand if the global minimum is not located inside it. Check the results posed by this method in Figure 4.16.

Discrete Membrane Shrinking Apart from the previously described variational methods, other strategies have been proposed using deformable surfaces with an indicator function as a result. One of them is the one presented by Esteve et al. [76], where an initial membrane is continuously deformed using a sequence of contraction operations that are executed sequentially. It has the advantage of not relying on distance functions, nor needing to compute neighbors' relations between points.

They formulate the α-shapes concept in a discretized voxel grid by changing the *eraser* object from a ball to a cube. The aim of the algorithm is to obtain the solid that would remain after *erasing* the voxel grid with cubes of different sizes. Larger cubes are used to reveal a crude volume, and sequentially smaller cubes are used to reveal finer details. Furthermore, it takes into account the possibility of having more than one component described in the working space.

Voxels are marked as *hard/soft*, according to whether they contain a point from P, or an *inside/outside/boundary* relative to the current membrane at a given iteration in the method.

The main algorithm consists of voxelizing the space, and marking as *hard* voxels those containing an input point. Then, a discrete membrane of face connected voxels is initialized as the 6 faces of the voxelization. Then, this membrane is iteratively contracted at locations having *soft* voxels, using *hard* voxels as a fix reference to restrict the shrinking process.

The shrinking of the discrete membrane is performed by using entities called the plates. A plate is a square of $n \times n$ voxels aligned with a plane defined by a coordinate in the reference frame (X, Y or Z). The front/back sides of the plate are the $n \times n$ voxels located in front/back of the plate. The lateral sides are the voxels located around the plate on the same axis-aligned plane. The lateral front/back sides are the voxels located on the lateral sides, but on the front/back plane. The Discrete Membrane (DM) Shrinking procedure uses 3 different operators for a plate of a given size:

- **Contraction:** A plate made of *soft* boundary voxels or outside voxels whose back-sides are made of outside voxels modifies the DM by converting boundary voxels that belong to the plate in outside voxels, and front, lateral and lateral front voxels of the plate to boundary voxels.

- **Undo Contraction:** This reverses the Contraction operation. This only happens when a set of Contraction operations lead to an *incursion.* Each *outside* voxel keeps track of its *generation*, i.e., the plate size that has converted it into an *outside* voxel. An *incursion* occurs when two outside voxels of different generations are on opposed faces of a boundary hard voxel.

- **Freezing:** This converts boundary voxels into *frozen* ones, which are *soft* voxels that are treated as *hard* voxels in future tests.

The algorithm starts with a large plate, and uses the 3 operations above to deform the DM. Then, at each iteration, the size of the plate is decreased, covering finer parts of the object.

Due to the use of a planar plate of voxels to deform the surface, the final shape tends to have flat regions of strong steeping in areas of low density of points. For this reason, a relaxation step is needed to smooth the resulting surface in these areas. Even with the smoothing step, the resulting surface has a steep look if the surface is extracted directly using marching cubes, since it is not an implicit but an indicator function, so a post-processing after surface extraction is needed for smoothing these features in order to obtain a visually pleasant surface.

Self-Organizing Map Methods commented on up to this point are based on an implicit formulation of the deformable surface. However, some authors have proposed methods to deform an explicit surface representation, despite the increasing level of complexity that requires keeping track of its evolution.

From this last class of methods, there is a set that adapts the Neural Network (NN) concept for surface reconstruction. Neural networks are grids of interconnected nodes which respond to a set of input signals, and which try to mimic the behavior of their biological counterparts. They are used in both supervised and unsupervised learning.

In an early project by Yu [212], a Kohonen's Self-Organizing Map (SOM) is used as a structure to recover the surface. The NN is arranged into a triangular mesh. Each neuron represents a vertex of a surface that deforms during the learning process of the SOM, so the weights of each cell are their coordinates in 3D space. At each step of the learning process, the input is a randomly chosen 3D point from the set. The point closer, in Euclidean distance, to that point is the *winner* point, and its weight and that of its neighbors is modified to approach this distance. The evolving surface following this approach tends to get stuck in concavities. In order to solve this, the edge swap operation [105] is used to disambiguate these cases. Moreover, the learning is performed at different sequential resolutions for the SOM mesh, letting the coarse meshes adapt to the global topology, with finer meshes representing the details. From one level in resolution to the next, triangles are divided into four by creating a new vertex at the middle of each edge. Note however that the topology of the initial mesh cannot change through the process.

Growing Cell Structure Following the above mentioned approach, we find the work of Ivrissimtzis et al.[110, 109]. The main differences between this work and that of Yu [212] is that in this case a Growing Cell Structure (GCS) is used. The GCS approach allows the NN to grow incrementally during the learning process instead of being static as in the

SOM method.

The main steps are similar to the ones presented by Yu [212]: a random point is the input to the network, and its closest point on the net along with its neighbors are moved towards the point. To speed up the nearest-neighbor search, the vertices of the network are stored in an octree structure that is updated when a vertex position changes.

As stated above, the GCS grows incrementally, so its topology must change after a certain number of learning iterations. The initial surface could be something simple, like a tetrahedron, and the method should have the ability to change the number of vertices, create boundaries and find handles on the surface. Vertices store a counter that represents the times it has been the *winner* in the learning steps. If a vertex has a large value for this counter, it is split using a vertex split operation, since it represents an area that is relevant for the surface described by the points, and more vertices should be used to represent it. Contrarily, the vertices having a low value in the counter, i.e., the most inactive ones, must be deleted with a half-edge collapse operation. Both the vertex split and the half-edge collapse operations are described by Hoppe et al.[105], and do not change the topology of a given mesh. However, the method also allows triangle removal and boundary merge operations, which create and merge boundaries respectively, thus changing the topology of the evolving mesh. Triangle removal is performed in very large triangles, which represent parts of the surface that are not correctly sampled given P. On the other hand, two boundaries are merged if they are close enough according to their Hausdorff distance.

Furthermore, two approaches are provided to improve the performance of the NN. Since the selection of the input point at each step is a random process, the neural mesh is not necessarily the same at any execution. Resulting models can contain errors in some cases, so the authors propose using the *ensembles* technique, which consists of executing the learning a given number of times and merging the results. The algorithm is then applied, and the different results are merged by using an averaged voxelized model. The implicit function inside this voxelized model is extracted and the learning starts again. Also the *forgetting* approach is used, which consists of coarsely voxelizing the interior of the moving mesh to delete the details. This approach usually improves the quality of the learning process in NNs.

Implicit SOM We have seen that approaches based on NNs have posed the surface to track in its explicit formulation. However, this technique can also be used to track an evolving implicit function modelled in a SOM, as presented by Yoon et al. [211].

The voxels of the regular grid discretizing the working space are, in this case, the nodes of the NN, which contain a scalar value. Then, the algorithm applies three sequential steps iteratively:

1. Generate training data: Using points with normals, samples are generated along the normal direction, both inward and outward, in order to create a sample of the signed distance function [43].

2. Competitive learning: The node v_i nearest to a training sample p_s updates its scalar value by $v_i = v_i + \alpha(I(p_i) - v_i)$, being the variable α a parameter governing the sensitivity of the training.

3. Propagation/smoothing: Note that the learning step only affects a subset of G, corresponding to the nodes nearest to the training data. In order to propagate the learning away from the data, a set of iterations of Laplacian smoothing is performed: $v_i = \lambda v_i + (1 - \lambda) mean(GN_6(v_i))$. The λ value is decreased at each iteration of the learning process.

After some iterations of the three steps above, the SOM converges. However, the authors noticed that the previously described process leads to an overfitting of the isosurface to the data. They further propose an overfitting control methodology, by selecting, at the competitive learning step, only one set of samples as training, and leaving the rest to validate the current I. After training, the validation set is compared to the values in the implicit function, and the errors are computed. If the errors do not differ too much from the ones in the previous iteration, the process is stopped.

Coarse-to-Fine Explicit There are other ways of evolving an explicit surface to fit a given point cloud. A good example is the work of Sharf et al. [190], where an initial closed surface placed inside the object is deformed following a scalar field representing the unsigned distance to the points. The main idea is to follow a coarse-to-fine approach, and force the surface to represent the global geometry before going for the detailed parts. The model is deformed until it is sufficiently close (according to a user's parameter) to P, and then an MLS projection operation is performed to project vertices to the point set. It also applies mesh optimization procedures at each iteration to ensure the quality of the mesh.

The deformable surface is initialized as a small sphere formed by an unstructured triangle mesh, placed in the interior of the point cloud. The placing of this seed surface is left to the user. The evolution moves the vertices in an outward normal direction, following the scalar field. This scalar field is made from the unsigned distances of each voxel to its closest point. Notice that it can be unsigned because the motion is guided by the outward normal direction, and the scalar field is only used to define the speed of the movement. The scalar function is represented inside an octree, and is made from a coarse RBF approximation using the method by Ohtake et al. [167] to define the unsigned zero level set, followed by the fast marching method proposed by Zhao et al. [216] for signing. The evolution resembles a balloon inflating inside the object, and different fronts or sets of points compete at the same time for evolution. A parameter accounting for surface tension is responsible for the coarse-to-fine behavior. It starts as a large value, and is incrementally released each time the evolution of a vertex is stopped. Also, when the tension is below a certain threshold, the front is subdivided. The model can arrive at an inactive state before all the target points are within a small given distance from the target points, so some mesh components are redefined as fronts by using a *wake up* procedure. This procedure consists of defining the set of unsatisfied vertices, i.e., those with a large distance to the points. A distance transform is computed using only those points, and the closest components in the mesh are activated as fronts. These fronts are subdivided and their tension is released, allowing further movement, which is guided now by the distance transform of the unsatisfied points.

At each iteration, the new positions of the vertices are computed using least squares for solving a constrained Laplacian system, following the approach by Sorkine and Cohen-Or [198]. When the vertices of the mesh are sufficiently close to the point set, the vertices are projected to the MLS surface defined by P.

4.5.6 Gradient Enforcement

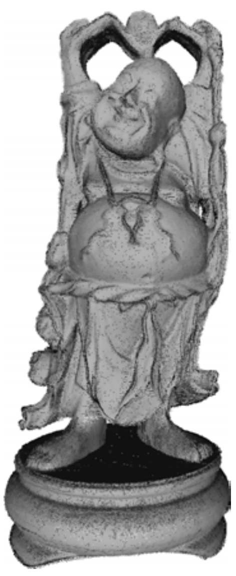
The methods falling in this category regard the set of input points with normals as a discrete set of samples of a gradient field over $\mathbb{R}^3$. This gradient field can be used to impose some restrictions on the implicit surface described by the points. Given its properties when casting the problem in this formulation (commented on in each method), the indicator function is normally selected to represent the surface of the object.

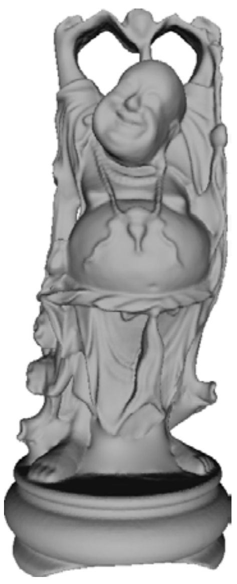
Luckily enough, all the methods in this section have an implementation provided by the authors, and a visual comparison containing all of them can be seen in Figure 4.17.

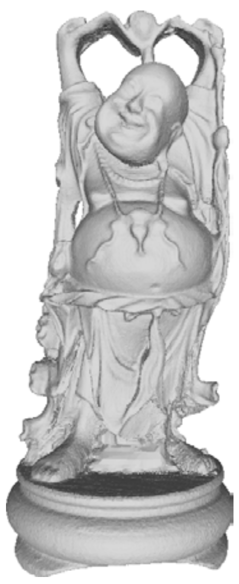
Fourier Transform The FFT method proposed by Kazhdan [117] was the first to exploit gradients for surface reconstruction. In his approach, the indicator function of the object is defined using the Fourier transform. Basically, the Fourier coefficients of the indicator function described by the points are computed, and the indicator function is finally recovered with its inverse transform.

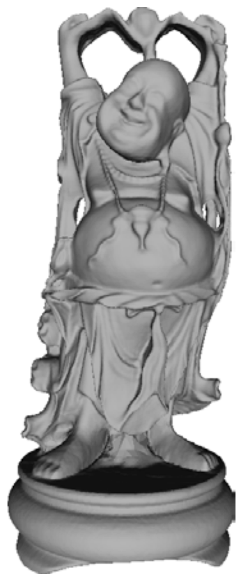
As with some of the methods in this category, it is based on Stoke's theorem, and more specifically, on the Divergence (a.k.a. Gauss) theorem, which allows the expression of a volume integral as a surface integral, thus providing a method for expressing the integral of a function over the interior of a region as an integral over its boundary:

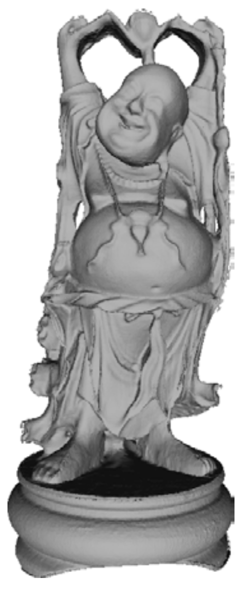
$$\int_M \nabla \vec{F}(p_x) dp_x = \int_{\partial M} \left\langle \vec{F}(p_x), n_x \right\rangle dp_x, \tag{4.83}$$


(a) Happy Buddha dataset


(b) FFT


(c) Wavelets


(d) Poisson


(e) Screened Poisson

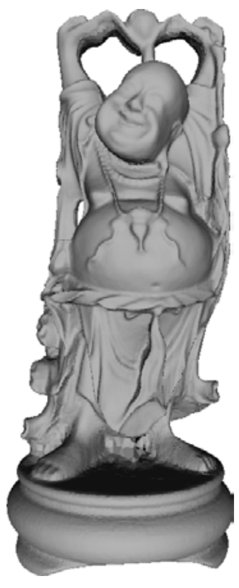

(f) SSD

Figure 4.17: Examples of the Gradient Enforcement methods applied to the Stanford Happy Buddha dataset. From left to right, top to bottom: input point set, FFT [117], Wavelets [153], Poisson [118], Screened Poisson [119] and Smooth Signed Distance (SSD) [41]. In all cases, the depth of the octree used is 10, except for the FFT method, which uses a fixed regular grid of size 500^3.

where M is a volume, $\vec{F}(p_x)$ defines a vector field, and $\nabla\vec{F}(p_x)$ its divergence. Obviously, with only the input points as samples on the surface, the previous integral can not be applied directly. Nevertheless, a Monte Carlo approximation can be computed using this discrete set:

$$\int_M \nabla\vec{F}(p_x)dp_x \approx \frac{|M|}{N}\sum_{i=1}^{|P|}\left\langle \vec{F}(p_x), n_x \right\rangle . \tag{4.84}$$

As previously stated, given a solid object M and its indicator function χ, the problem is to find the associated Fourier coefficients. Those coefficients being $\hat{\chi}$, they can be obtained using a volume integral. However, and by means of using the Divergence theorem, the volume integral can be cast to a surface integral:

$$\hat{\chi}(p_f) = \int_{[0,1]^3}\chi(p_x)e^{-2\pi i\langle p_x,p_f\rangle} = \int_M e^{-i\langle p_f,p_x\rangle}dp_x = \int_{\partial M}\left\langle \vec{F}(p_x), \vec{N}(p_x)\right\rangle dp_x. \tag{4.85}$$

Note that while throughout this chapter we used i as an index, here it represents the imaginary unit. Additionally, the surface integral can be discretely defined by the input oriented point cloud using the Monte Carlo approximation:

$$\hat{\chi}(p_f) = \frac{1}{|P|}\sum_{j=1}^{|P|}\left\langle \vec{F}(p_x), n_x \right\rangle . \tag{4.86}$$

Now, $\vec{F}$ has to be a function whose divergence $\nabla\vec{F}$ is equal to the complex exponentials, to agree with Equation (4.83). From the many possibilities following this equality, the authors propose using a function that, when converted to its indicator function, rotates according to the normals of the input points. Being $p_x = (x_x, y_x, z_x) \in P$ and $p_f = (x_f, y_f, z_f) \in \chi$:

$$\vec{F}(p_f)(p_x) = \begin{pmatrix} \frac{i\cdot x_f}{x_f^2+y_f^2+z_f^2}e^{-i\cdot(x_f x_x + y_f y_x + z_f z_x} \\ \frac{i\cdot y_f}{x_f^2+y_f^2+z_f^2}e^{-i\cdot(x_f x_x + y_f y_x + z_f z_x} \\ \frac{i\cdot z_f}{x_f^2+y_f^2+z_f^2}e^{-i\cdot(x_f x_x + y_f y_x + z_f z_x} \end{pmatrix} . \tag{4.87}$$

Given these definitions, one is able to compute the Fourier Coefficients up to an additive and multiplicative constant. However, a single coefficient requires a summation over all input points.

In order to avoid this complexity, the authors propose casting the problem as a convolution. If the point cloud is splatted on a regular grid, it results in a *gradient field*, containing only values at the positions of P. The Fourier coefficients of this gradient field $\hat{\vec{N}}$ and those from the indicator function are related:

$$\hat{\chi}_M(p_f) = \frac{i}{\|p_f\|^2}\left\langle \hat{\vec{N}}(p_f), p_f \right\rangle . \tag{4.88}$$

Thus, the simplified computation consists of splatting the normals onto the voxel grid, obtaining $\hat{\vec{N}}$, and then using the fast Fourier transform to convolve $\hat{\vec{N}}$ with the following filter:

$$\hat{\vec{F}}(p_f) = \frac{i(x_f, y_f, z_f)}{(x_f^2, y_f^2, z_f^2)}. \tag{4.89}$$

Notice that the recovered indicator function, as previously observed, has an unknown additive factor (because the function is undefined at $p_f = (0,0,0)$) and an unknown multiplicative factor (the surface area, $|M|$ term in Equation 4.84, is unknown). For this reason, in order to obtain a valid isovalue to extract the surface from the recovered pseudo-indicator function, the values of the characteristic function at the input points are recovered, and an average of these values is used. This is motivated by the fact that points P should lay on, or close to, the surface of interest.

So far, the point set has been assumed to have a regular sampling, so the Monte Carlo approximation could be applied directly. However, since real datasets usually have a non-uniform sampling, the authors propose weighting each point contribution to the approximation by a factor describing its sampling density. In order to do so, they propose a heuristic consisting of splatting a Gaussian on each point from P in the volumetric grid, and merging them together. Then, the weight for each point is set as the inverse value resulting in its position on the grid.

There are some methods that try to improve the original method, including new heuristics. This is the case of the method proposed in Schall et al. [184], were the memory limitations of FFT reconstruction are partially solved by using a partition of unity approach (see Section 4.5.3). The space is subdivided using an adaptive octree, and the FFT reconstruction algorithm is applied on each cell. An explicit representation is constructed and the Hausdorff distance from the points falling in that cell to the reconstructed mesh is computed. If this distance is above a threshold, the cell is further subdivided and the process continues. In order to avoid badly reconstructed areas at the borders of the leaf octree cell, the points taken into account to compute a local reconstruction for a cell are those falling inside the area defined by doubling the cell size. Local contributions to the indicator function are blended together using Gaussian weighting.

Wavelets The main drawback of FFT reconstruction is that Fourier coefficients are globally supported functions, which makes them complex to evaluate on large datasets. As a proposal to overcome this problem, Manson et al. [153] substitute the Fourier coefficients with a hierarchy of compactly supported wavelet functions. The algorithm has two parameters modifying the tradeoff between the accuracy of the reconstruction and the computational complexity: the wavelet basis and its support size. Smoother wavelet functions lead to smoother surfaces, but larger complexity, while large support size also leads to a better approximation, but at the expense of larger complexity. The reconstruction process may also be executed in a streaming fashion, keeping in memory small parts of the model at the same time.

Wavelet expansion provides simultaneous localization in both frequency and spatial domain, contrary to the Fourier expansion, which only refers to the frequency domain. We briefly introduce the main ideas behind wavelets, describing them first in 1D for simplicity, and then extending them to $3D$. In order to briefly introduce $1D$ wavelets and their associated transform, we must first talk about *scaling* functions. A scaling function has the following form:

$$\varphi_{j,k} = 2^{j/2}\varphi(2^j t - k), \tag{4.90}$$

where j and k are its parameters. The first one, j, accounts for the **scale** of the function: the higher the j value, the higher the frequency of the function, and viceversa. This parameter has a similar meaning to the parameter n, which represents the frequency in the Fourier expansion formula: $\varphi_n(t) = e^{-j2\pi nt}$. On the other hand, parameter k represents the **position** of the function, that is, its shift. As already mentioned, this is the spatial part missing in the Fourier transform. If function $\varphi_{j,k}$ is a *scaling function*, they are a family of basis functions, and therefore this means that any given function $f(x)$ can be expressed by a linear combination of them:

$$f(x) = \sum_j \sum_k c_{j,k}\varphi_{j,k}(x). \tag{4.91}$$

Each of the scaling functions define a function space U_j. If we name W_j the difference between the two function spaces U_{j+1} and U_j, we get that W_j contains the functions representable in U_{j+1} but not on U_j, because of its coarser resolution. In the same way when the function space U_j is spanned by functions φ_j, the function space W_j can also be spanned by a set of functions called *wavelets*:

$$\psi_{j,k} = 2^{j/2}\psi(2^j x - k). \tag{4.92}$$

Given the concept of scaling and wavelet function, a wavelet expansion is defined as follows:

$$f(x) = \sum_k a_{j_0,k}\varphi_{j_0,k}(x) + \sum_{j=j_0}^{\text{inf}} \sum_k d_{j,k}\psi_{j,k}(x), \tag{4.93}$$

where the first term represents the *approximation* of the function at the first scale level j_0, which is a linear combination of scaling functions $\varphi_{j_0,k}(t)$, and the second term represents its *details*, by using a linear combination of wavelet functions from higher scales. Following this nomenclature, $a_{j_0,k}$ are the approximation coefficients, while $d_{j,k}$ are the detail coefficients.

Notice that $\psi_{j,k}$ and $\varphi_{j,k}$ are the same, so we can express Equation (4.93) as:

$$f(x) = \sum_k c_{j_0,k}\psi_{j_0,k}(x) + \sum_{j=j_0}^{\text{inf}} \sum_k d_{j,k}\psi_{j,k}(x), \tag{4.94}$$

where each coefficient $c_{j,k}$ is given by:

$$c_{j,k} = \int_{\mathbb{R}^3} f(x)\psi_{j,k}(x)dx. \tag{4.95}$$

The extension to 3D is based on the displacement k. The displacement must be performed in each of the possible directions of k. Thus, keeping in mind that we pretend to do a Discrete Wavelet Transform (DWT), displacement k should be applied in 8 possible directions. These 8 directions, which coincide with the set of vertices of a cube in $[0,1]^3$, are the ones where we can apply Equation (4.94). If we mark this displacement with a superindex, the formula is extended as follows:

$$c^e_{j,k} = \int_{\mathbb{R}^3} f(x)\psi_{j,k}(x)dx, \tag{4.96}$$

where e is the mentioned direction.

Back in the surface reconstruction method, our function $f(x)$ is the indicator function χ of the volume M, so substituting it in Equation (4.96), the formulation can be derived in the following form:

$$\begin{aligned} c^e_{j,k} = 2^{3j/2} \int_M \quad & \psi^{e_1}(2^j x_1 - k_1) \\ & \psi^{e_2}(2^j x_2 - k_2) \\ & \psi^{e_3}(2^j x_3 - k_3)dx. \end{aligned} \tag{4.97}$$

As in the previous FFT proposition above, we do not have samples from the interior of the solid defined by χ, only samples from its boundary. Again, the solution consists of using the divergence theorem to represent the volume integral as a surface integral, following Equation (4.83). Also using a Monte Carlo approximation, the coefficients can be expressed in terms of the input point samples:

$$c^e_{j,k} = 2^{3j/2} \sum_i F^{\vec{e}}_{j,k}(p_x), \vec{n}_{p_x} a_i, \tag{4.98}$$

where a_i is an approximation of the differential surface area at p_i. This a_i approximation is obtained from an octree representation where the points are splatted, and is shown in the following:

$$a_i = 2^{-2d_i}/l, \tag{4.99}$$

where d_i is the depth of the octree cell, and l is the number of points falling in that leaf.

In order to fulfill the equality in Equation (4.83), and again as in the FFT case, the vector-valued function $F^{\vec{e}}_{j,k}(p_x)$ has to accomplish:

$$\nabla F^{\vec{e}}_{j,k}(p_x) = \psi^{e_1}(2^j x_1 - k_1)\psi^{e_2}(2^j x_2 - k_2)\psi^{e_3}(2^j x_3 - k_3). \tag{4.100}$$

The authors provide a way of computing $F^{\vec{e}}_{j,k}(p_x)$ for each compactly supported wavelet. Compact support computation of the coefficients provides less computational effort than the FFT method, where each coefficient is described by all the input points. Notice that, despite the wavelet coefficients having compact support, the coefficient $c^{(0,0,0,)}_{0,k}$, corresponding to the approximation scaling function, does require a summation over all the input points.

It is worth notice that the selection of the wavelet function to use affects the quality and the speed of the reconstruction. Obviously, the more compact support the wavelet has, the faster the method is in computing the coefficients. On the contrary, the more support, the more quality of the final χ. For this reason, and in order to maintain a fast reconstruction, the authors propose using compactly supported wavelet functions, and then smoothing the output χ using a method optimized for octrees before obtaining the surface from it.

Their implementation provides a way to compute the indicator function in a streaming fashion, allowing the recovery of models that do not fit in memory by processing small parts one at a time. First, the input points are sorted following a given direction, e.g., the Z of the reference frame. Then, a low resolution version of χ up to some octree depth d_m is built in memory using the above mentioned methodology. Next, the method is applied in slices, advancing in the previously defined direction. The coefficients $c^e_{j,k}$ of χ for $d_m < j < d_{max}$ are built, d_{max} being the maximum depth we want to achieve. At each slice, points are inserted and the tree is refined if its resulting coefficient has a non-zero value.

Poisson The most representative method of this part of the classification is the Poisson method of [118]. Having the gradient of the indicator function being defined by the input points with normals, one can cast the surface reconstruction process as the problem of finding the indicator function whose gradient approximates the vector field $\vec{N}$ defined by the input oriented samples:

$$min_{\chi} \left\| \nabla\chi - \vec{N} \right\| . \tag{4.101}$$

By means of applying the divergence operator, this can be cast to a Poisson problem:

$$\Delta\chi = \nabla\nabla\chi = \nabla\vec{N}, \tag{4.102}$$

that is, compute the scalar function χ whose divergence of gradients, i.e., its Laplacian: $\nabla\nabla$, equals the divergence of the vector field $\vec{N}$ defined by the input samples, which we already consider as an approximation of the gradient of χ. As noticed by the authors, the relationship between $\nabla\chi$ and $\vec{N}$ is not direct. The indicator function χ is a piecewise constant function, so its gradient is not well defined at the interface between $\chi(p_x) = 1$

and $\chi(p_x) = 0$. For this reason, instead of assuming the perfect χ, they assume to be working with its *smoothed* version. They use a Gaussian smoothing function ϕ to obtain the smoothed indicator $\chi * \phi$, leading to the following approximation:

$$\nabla(\chi * \phi)(p_x) \approx \sum_{p_i \in P} area(p_i)\phi_{p_i}(p_x)n_i \equiv \vec{N}(p_x). \tag{4.103}$$

The presented equality mainly verges from the divergence theorem, and a Monte Carlo approximation of the surface integral using the point samples weighted by an approximation of the area $area(p)$ corresponding to that patch. This approximation of the surface area at a given point is given by a procedure similar to the one executed in the FFT method: splatting + convolution of the points using a Gaussian kernel. This procedure produces a set of implicit weights $w(p_x)$, which are directly considered as an approximation of $area(p_i)$.

When building this approximated vector field $\vec{N}$, the requirements of being accurate near the surface is used to build it into an octree representation. As usual, this representation provides an alleviation in the memory requirements to perform the computations. Each node o forming the octree has an associated function $\phi_o(q)$ centered on it and scaled to fit its extension:

$$\phi_o(p_x) = \phi\left(\frac{p_x - centroid(o)}{w(o)}\right)\frac{1}{w(o)^3}. \tag{4.104}$$

In this way, functions of deeper nodes are associated with finer and more precise representations of the surface. The contribution of the functions of each node is distributed to the eight neighbors on the same octree level using trilinear interpolation. As previously mentioned, the contribution of each point is weighted by their approximation of the area they cover, so the $\vec{N}$ field computation remains as follows:

$$\vec{N}(p_x) = \sum_{p_i \in P} \frac{1}{w(p_i)} \sum_{ON_{8,odepth(p_i)}(p_i)} \alpha_{o,p_i}\phi_o(p_i), \tag{4.105}$$

where α are trilinear interpolation weights. Once the vector field is constructed, the divergence operator is applied in order to obtain $\nabla\vec{N}$.

All the information needed to solve the problem $\Delta\chi = \nabla\vec{N}$ is now available. If we write the system with a matrix L so that Lx returns the dot product of the Laplacian with each ϕ_o, the minimization can be written as follows:

$$\min_{x \in \mathbb{R}^{|O|}} \|Lx - M\|^2, \tag{4.106}$$

being $|O|$ the number of nodes in the octree, and M is an $|O|$-dimensional vector so that $m_o = \langle \nabla\vec{N}, \phi_o \rangle$. In order to solve this system of equations, the sparseness and symmetry of L (provided by the compact support and symmetry of ϕ) are taken into account. Furthermore, the inherent multiresolution of the octree and, consequently of ϕ_o, is exploited to use a multigrid approach when solving Equation (4.106). Finally, and in order to limit the amount of memory required to solve the system, the authors propose using a block Gauss-Seidel solver for depths larger than a user's defined threshold.

The final extraction of the isosurface follows the same proposal as FFT, i.e., find the isovalue as the mean of the recovered indicator function at the input points. This procedure is again motivated by the fact that χ is only recovered up to an unknown multiplicative factor.

As the authors demonstrate in the appendix, the results obtained by the Poisson method and the FFT method are equivalent, but the Poisson method requires far less computational effort and memory requirements to be computed, allowing the method to cope with larger datasets. Furthermore, a streaming out-of-core implementation to ameliorate its complexity under large datasets has been proposed [31], along with further GPU implementations [218] to speed up its execution.

However, a major improvement in this technique has been presented in the Screened Poisson method [119]. Motivated by the oversmoothing that the original method provides under some configurations, this extension accounts for a user's parameter that imposes a soft constraint on the resulting isosurface to pass through the input points, i.e., the method behaves closer to an interpolative approach. Knowing that in the ideal indicator function, the energy to minimize becomes then:

$$E(\chi) = \int \left\| \vec{N}(p_x) - \nabla\chi(p_x) \right\|^2 dp_x + \lambda \sum_{p_i \in P} \|\chi(p_i) - 0\|^2 , \tag{4.107}$$

where the first term is the same as in the original implementation, Equation (4.101), and the second imposes a value-fitting constraint, λ being the user's parameter guiding the importance of this interpolatory weighting. As further improvements, the approximation of χ is done in this case using B-splines as basis functions, the screened Poisson changes boundary conditions from Dirichlet to Neumann, and alleviates the algorithmic complexity of the solver.

Smooth Signed Distance Inspired by the fair applicability of the Poisson method, the proposal by Calakli and Taubin [41] aims at recovering a signed distance function instead of an indicator one. Taking again the points with normals as samples of the surface and its gradient, they propose a variational formulation minimizing in a least-squares sense the signed distance function induced by these oriented points:

$$E(f) = \lambda_0 \left[\frac{1}{N} \sum_{1}^{N} f(p_i)^2 \right] + \lambda_1 \left[\frac{1}{N} \sum_{1}^{N} \|\nabla f(p_i) - n_i\|^2 \right] + \lambda_2 \left[\frac{1}{M} \int_M \|H(x)\|^2 \, dx \right], \tag{4.108}$$

where each term to minimize is under square brackets and the λ_x weights are used to tune their weights in the minimization. As we can see, the first and second terms account for the minimization of discrepancies between values of the function at points, which should be zero, and between gradients of the function and normals at points respectively. The final term accounts for regularization, where $H(x)$ stands for the Hessian matrix of f at x, integrated over the whole volume M.

This minimization is solved in an octree structure. More precisely, for each cell in the octree, a trilinear interpolation of $f(x)$ is used, while finite differences are used to compute both $\nabla f(x)$ and $H(x)$. By fixing a basis function, the whole minimization can be expressed in terms of the parameter vector F, which reduces to a minimization of the form $E(F) = F^T AF - 2b^T F + C$, having a global minimum at $AF = b$.

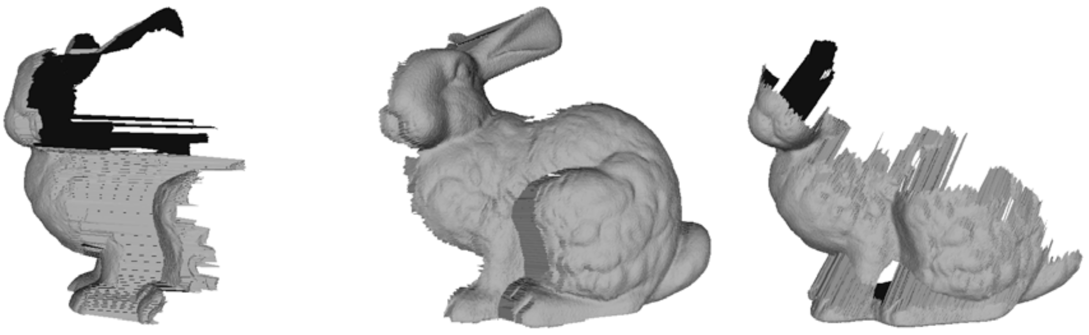
4.5.7 Integration

As the title of this section suggests, the methods in this category are similar in nature to those in Section 4.4.1.3, but provide an approximate result as output. Thus, local 2D connectivity is again assumed to be known for different *shots* or sets of points, and these local reconstructions are merged together into a single model.

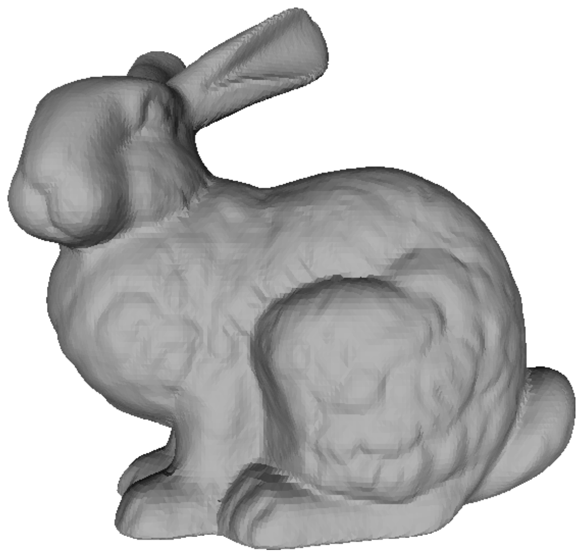
Volumetric Range Image Processing The most widely known and used approach is the Volumetric Range Image Processing (VRIP) by Curless and Levoy [60]. It provides a simple idea for merging local scans into a single voxel grid, by updating a distance function one range image at a time. Each range image is transformed to a distance function and added to the global distance function, following a weighted addition scheme.

More specifically, the global signed distance function I^D is the signed distance from each voxel v_i to the nearest range measure along the line-of-sight of the sensor. Each range image is transformed to a local signed distance function $I_i^{LD}(v_i)$, and a local weight function $I_i^{LW}(v_i)$ is also computed for each of them. Then, each single distance and weight function is placed in their correspondent positions in the global frame, since, as usual, we assume the scans are registered in a global frame. Each voxel in $I^D(v_i)$ has an accumulated distance and an accumulated weight value $I^W(v_i)$, computed with the following simple additive scheme:

$$\begin{gathered} I_i^D(v_i) = \frac{I_i^W(v_i) I_i^D(v_i) + I_i^{LW}(v_i) I_i^{LD}(v_i)}{I_i^W(v_i) + I_i^{LW}(v_i)} \\ I_i^W(v_i) = I_i^W(v_i) + I_i^{LW}(v_i). \end{gathered} \tag{4.109}$$



(a) Sample of range scans



(b) VRIP

Figure 4.18: Sample of the VRIP [60]. The top row shows a sample of 3 range scans (from a total of 10) forming the Stanford Bunny dataset, while the second row shows the result of the algorithm.

Weights are used to represent the uncertainty of the measurements. For example, a small weight should be given to points on the boundaries of the range image, since they are more likely to be erroneous.

Range images are transformed to local signed distance functions by using ray casting. In order to alleviate the computational effort required to create these local functions, the distance is only computed at a given range before and after the local surface, i.e., distance is only defined near the surface. The authors also propose a hole filling procedure, consisting of carving the empty outer space using lines-of-sight, and then considering the interface between carved and non-carved space as surface. For a sample of the behaviour of this algorithm, see Figure 4.18.

Scale Space Fusion Following the volumetric idea of VRIP, the proposal by Fuhrmann and Goesele [83] extends the method to work in scale space. Thus, the scale of each scan (i.e., detail or resolution) is taken into account while merging them, thus enforcing the preservation of fine details.

In this case, each of the levels in an octree structure represents a scale value. In fact, finer levels in an octree naturally coincide with higher resolution in space. Thus, having into account the footprint of a triangle, namely its size, its contribution to the signed distance is added at its corresponding scale.

Then, starting with the coarser level, the voxel value of a lower scale is combined to that in the higher scale by interpolation of its distance/weight. At a second step, a fine-to-coarse traversal is performed to retain for each duplicated voxel at different scales just the one containing finer information.

Having a set of voxels with a given distance value, voxel positions are used to create a Delaunay triangulation. By doing so, for each required query point, the distance function can be retrieved by linear interpolation inside the corresponding tetrahedra. This arbitrary evaluation of the distance function allows using the conventional surface meshing approaches to extract the final surface.

Lifted Constrained Delaunay In the method proposed by Salman and Yvinec [180] the 2D triangulation of the points in their local image plane is used as the approximation. That is, it is considered as a local range image, as in the cases above. However, the global merging of these local contributions is not implicit.

The approach is devised to work with the output of a feature-based computer vision 3D reconstruction system. The processing starts with the gathering of the points themselves. The points are first tracked through the sequence, and a point is considered to be the same as another if its Euclidean distance is below a threshold. Then, the resulting points are filtered by using two criteria. The first one takes into account the distance of the nearest neighbors of a point, discarding it if the distance is too large (threshold). The second one considers the baseline of the views that have generated that point. A cone with its apex at the point and containing all the input cameras is computed. If the cone's angle is small, it is considered an outlier. Finally, a smoothing step is computed by fitting a quadric surface to the k-nearest neighbors of a point, which is then projected to this surface.

Once the point cloud is built, the surface can be extracted. This algorithm relies on the 2D planes to do most of the triangulation work. First, contours of the input 2D images are extracted, and the projections of the tracked points that lie near a contour pixel are marked as contour points. Then, connectivity between these points following the contours is built, taking into account the avoidance of finding intersections between edges of the resulting planar graph. Then, a 2D Delaunay triangulation constrained by these points/edges is computed for each frame using the 2D projections of the points seen from each view. The 2D triangles are then lifted to 3D, where they form a partially-disconnected, and possibly self-intersecting, set of 3D triangles.

Not all the triangles of this set lie on the surface, so they have to be further refined. Three filtering steps are applied:

1. Line-of-sight constraint: Triangles in conflict, i.e., intersected, by n lines-of-sight are removed.

2. Triangles seen by glancing lines-of-sight: Triangles are removed when their normal defines a large angle with the line-of-sight.

3. Shape criteria: It filters all triangles having a large ratio between the radius of its circumcircle and that of its shortest edge.

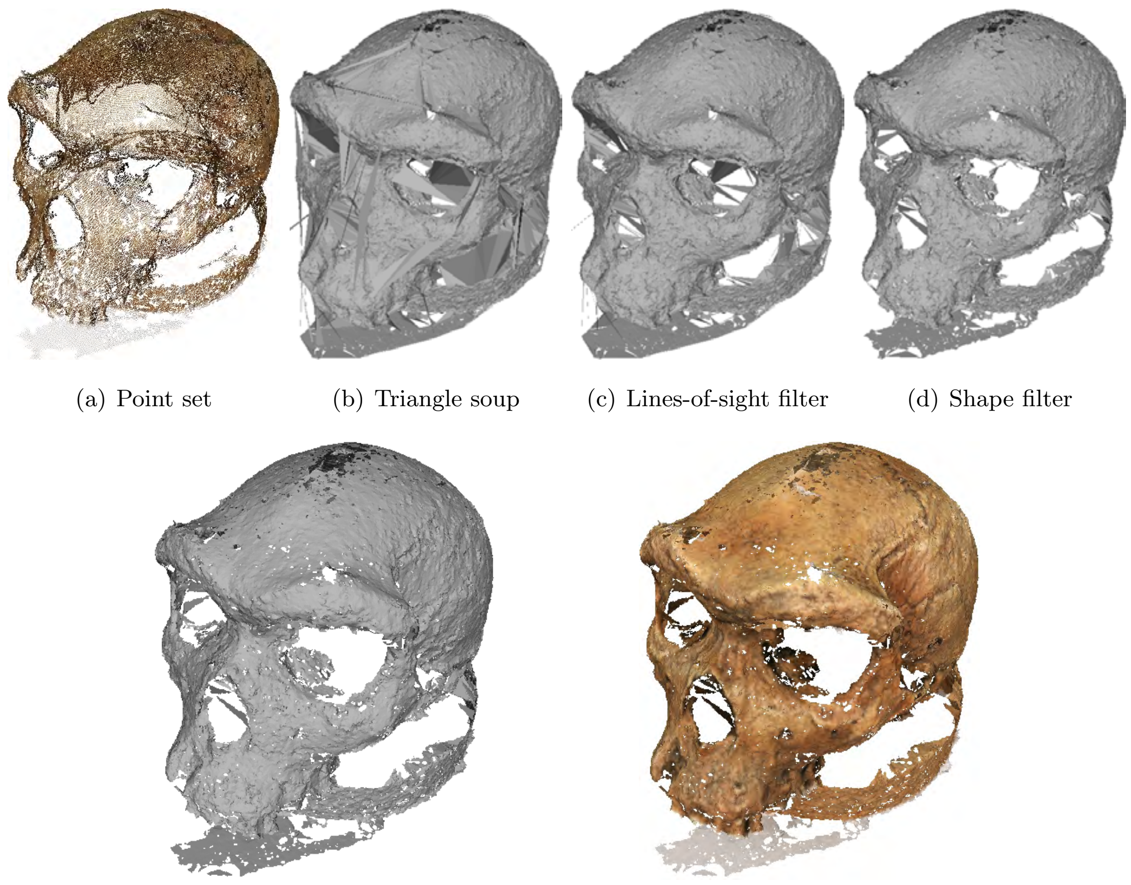
4. Photoconsistency criteria: A triangle is photoconsistent if its projections to all images where their vertices are visible correspond to similar image regions, taking the NCC as the similarity measure. This last criteria is only applicable to triangles with a relatively large area, enough to contain relevant texture information.

Once the triangle set has been filtered, they approximate the overall shape of the object, but they lack a globally coherent connectivity, i.e., they do not form a single mesh. For this reason, this set is only used as input for the surface mesh generation algorithm described in Boissonnat and Oudot [30]. Since the algorithm only needs an *oracle* structure that, given a line segment, detects whether it intersects the surface, the triangle soup is used directly for this purpose. That is, the intersections between the query segment and the triangles is computed, and the average of these intersections is returned as the intersection point for the query.

Figure 4.19 exhibits an overview of the different filtering stages of the process, along with the resulting surface, when applied to a multi-view stereo dataset of a skull object. Note however, that, in this case, constraining the triangulation to the apparent contours in the image makes little sense, since the object does not have any sharp edges.

4.5.8 Local Primitives

Finally, the approaches in this section are based on computing the surface using local primitives around the input points that are later used to build a global surface. It is worth mentioning that these approaches do not perform the merging of the local primitives in a global framework, such as using a global implicit function as some of the RBFs' cases.



(a) Point set (b) Triangle soup (c) Lines-of-sight filter (d) Shape filter

(e) Surface mesh (f) Textured surface mesh

Figure 4.19: Lifted Constrained Delaunay method [180]. Figure (a) shows the point cloud. The initial triangle soup made by joining the constrained 2D triangulations in 3D is depicted in (b). This initial triangle soup is filtered using lines-of-sight (c) and shape constraints (d). This last triangle soup (d) is used to obtain the final surface through Delaunay refinement, resulting in the surface in (e) - with its textured version in (f).

Instead, the merging of these primitives is also performed locally using a close vicinity around them following an explicit approach.

Spherical Cover This method, proposed by Ohtake et al. [168], is based on covering the input points using spheres, and then connecting some auxiliary points in these spheres to create the final surface mesh. By allowing the user to tune the accuracy of this spherical cover, it provides the creation of surfaces of multiple resolution with the same point set.

In a first step, each point is assigned an unoriented normal through PCA [104], along with a measure of density consisting of the distance to its k-nearest neighbors.

Then, a spherical cover of the point set is constructed by selecting at each iteration a non-covered point p_i as the center of the sphere. Next, the radius is optimized locally, along with an auxiliary point inside the sphere. The points falling inside the optimized sphere are projected onto the local tangent plane at p_i, and the points lying inside their convex hull in 2D are marked as covered. The process continues iteratively until there are no more points to cover. Finally, the auxiliary points are joined in triplets to form triangles if their corresponding spheres intersect.

Obviously, the key step is to find the optimal radius and auxiliary point for each sphere. The function to minimize is the following:

$$E_Q(p_i, r, p_a) = \sum_j w_j \cdot \phi_{i,r_\phi}(\|p_j - p_i\|)(n_i(p_a - p_i))^2, \tag{4.110}$$

where p_a is the auxiliary point, and r the radius of the sphere. In fact, the influence radius r_ϕ of the spline RBF ϕ plays an important role in the reconstruction accuracy, and is given proportional to r as $r_\phi = T_q r$, T_q being a user's parameter. Assuming r is known, the only parameter to optimize is the auxiliary point p_x, which is obtained by solving a system of linear equations. Thus, in order to optimize for both r and p_a, the final minimization uses the following equation:

$$E_r(r) = \frac{1}{d}\sqrt{E_Q(p_i, r, p_{a_{min}})}, \tag{4.111}$$

d being the diagonal of the bounding box enclosing P. Thus, several values of r are tested in order to obtain a user defined error $E_r = e$, solved using a bisection method.

Finally, after joining auxiliary points to form triangles, a procedure forcing the extraction of a manifold surface and some filtering steps are needed in order to recover a coherent mesh. Figure 4.20 shows the results obtained by this algorithm, and how the method allows for different complexities of the surface to be retrieved by changing a user's parameter.

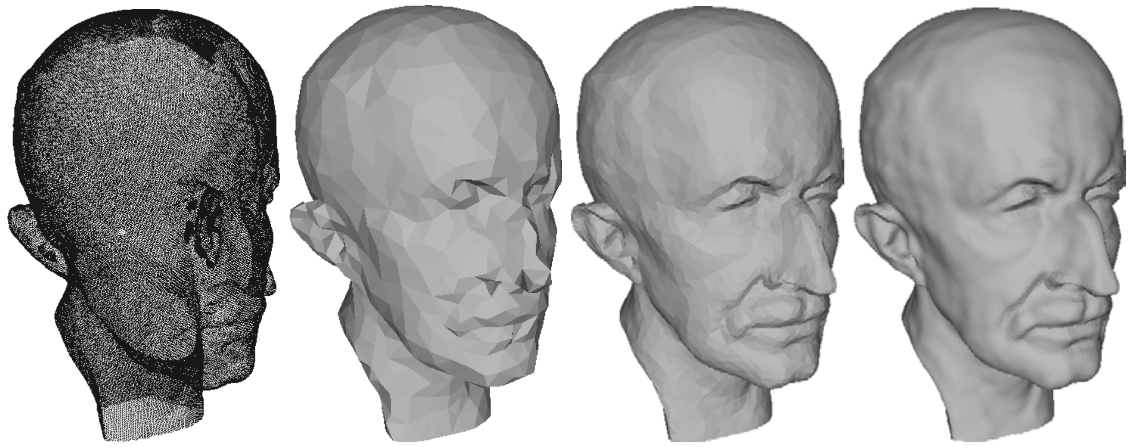


Figure 4.20: Spherical Cover method [168] applied on the Max Planck bust dataset (leftmost figure). The value of the parameter guiding the approximation quality (T_q in the original article) is increased from left to right, leading to an increasing level of complexity for the resulting surface mesh.

4.6 Conclusions

We have presented a review of the available methods to tackle the problem of surface reconstruction from an unorganized cloud of points. The different contributions in this field have been classified by the nature of the resulting surface according to the input point cloud. This selection is not arbitrary, and accounts for the further processing to be applied after processing. Of course, for visualization purposes, all methods serve as long as they provide a correct representation of the object. However, there are some cases where it is important to generate an interpolative surface passing through the input points. To give some examples, points may be known to be part of the true surface, i.e., the data is regarded as ideal, or they may have some associated information that may be required in further processing, as is the case of points reconstructed from a computer vision 3D reconstruction system, where each point contains which views from the original sequence of images have generated it. Besides, it has also been shown that it is easier to talk about the correctness of a method (i.e., propose theoretical guarantees) if the vertices of the resulting surface are part of the input points.

Additionally, methods inside each category have been divided according to common procedures and methodologies, despite that they may use common concepts or data structures. It has been our intention to provide a self-contained comprehensive review of the main procedures and steps conforming the algorithms. A common nomenclature has been used to help the reader relate the common procedures and structures used across the methods.

We have seen that the main concern of surface reconstruction is to find a heuristic able to solve this inherently ill-posed problem. A common approach is to rely on the sampling concept as the main indicator of where the surface is. The main property of sampling is its density or sparseness. Densely sampled areas are regarded by the majority of methods as indicators of the presence of a surface patch. However, sampling does not need to be regular over the entire surface, since smooth areas do not need to be described by many samples, as opposed to sharp features of the object or scene, which need a more detailed sampling to be represented. We can then conclude that a good sampling of a surface is highly dependant on its curvature, and this concept is also exploited by many methods.

While most approximation-based approaches provide an implicit noise correction to some extent, they rarely cope with outliers. Because of their global nature, methods such as Poisson reconstruction [118] achieve resilience to small quantities of outliers, but fail with a large number of them. Only a few methods exhibit robustness to both noise and outliers, and more importantly, very few of them are specifically designed to tackle both problems at the same time.

It should be noted that an important requirement of many surface reconstruction methods is knowing the normals at each input point. Since not all scanning systems provide normals, they must be computed prior to using one of these surface reconstruction methods. However, computing the normals is also an ill-posed problem, since estimating the normals requires inferring the surface locally. While some of the presented methods have tried to compute normals (i.e., local tangent planes) during the surface reconstruction process [104, 169, 8], it is worth noticing that all of them further require another heuristic to achieve a globally consistent orientation for all these normals. This consistent orientation is needed in order to be able to infer from these primitives a signed distance field, or a notion of inside/outside the object.

We also observe that the literature is considerably thinner on the question of robustness to outliers. While some interpolation-based methods are able to deal with outliers to some extent, they are not able to deal with noise. Thus, devising methods that are simultaneously resilient to both noise and outliers and that do not depend on a priori knowledge (such as normals) is direly needed. Our end goal in this thesis is to devise more general methods applicable to inputs coming from a wider variety of measurement devices and processes.

Another important, but commonly overlooked, problem is the reconstruction of surfaces with boundaries. Real datasets, not gathered in a laboratory environment, do not normally represent a simple closed object, but a possibly large, arbitrary scene. It is then usual to find bounded surfaces, and the watertight assumption is too restrictive in those cases. We have seen that a majority of methods assume watertightness to further restrict the computation of the surface. Consequently, these methods cannot be applied directly to real (e.g., outdoor) datasets, at least not without a post-processing to detect such boundaries.

Finally, a fair comparison of the methods presented is difficult to achieve because of the lack of implementation of the majority of them and also the parameter tweaking required for each method to achieve its best performance with a given dataset. A first benchmark solution has recently been presented by Berger et al. [22], where they present a framework for quality assessment in surface reconstruction approaches. In their article, they only surveyed the behaviour of some approximation-based methods, obviating the available implicit based methods. Furthermore, while the benchmark is designed towards testing the methods under different noise conditions, they do not take into account outliers. Thus, it would also be interesting to extend the qualitative review of this work to also allow for interpolation-based methods, and study further the behaviour of the methods with outliers present in the data.

Table 4.3: Surface reconstruction methods classification, interpolation based. Columns refer to the ability of the method to deal with boundaries, noise, outliers, if it provides theoretical guaranties, if it uses additional information rather than the point cloud and its complexity overview, if provided by the authors.

				Properties					
			Methods	**Boundaries**	**Noise**	**Outliers**	**Guaranties**	**Additional Info**[1]	**Complexity**[2]
Interpolation Based	**Surface Oriented**	**Delaunay Triangle Selection**	α-shape [72, 98]	Y	N	N	N	-	$O(n^2)$
			Normalized mesh [15]	N	N	N	Y	-	$O(n^4)$
			Crust [10]	N	N	N	Y	-	$O(n)$
			Cocone [12, 62, 63]	Y	N	N	Y	-	-
			Gabriel [171]	Y	N	N	Y	-	$O(n^2)$
			Umbrella Filter [3]	N	N	N	N	-	-
			Flow Complex [88]	Y	N	N	N	-	-
		Surface Growing	Boissonnat's Greedy [28]	N	N	N	N	-	$O(n \log n)$
			Graph Greedy [156]	Y	N	N	N	-	-
			Ball-Pivoting [23]	Y	N	N	N	-	$O(n)$
			Spiraling Edge [58]	Y	N	N	N	N	-
			Lower Dimensional Delaunay [92]	Y	N	N	Y	-	-
			Smooth Greedy [54, 129, 142, 108, 140]	Y	N	N	N	-	$O(n^2)$
			Medial Scaffold Transform [48]	Y	Y	N	N	-	-
		Integration	Zipper [204]	Y	N	N	N	LC, SP	-
			Venn Zipper [199]	N	Y	N	N	LC, SP	-
	Volume Oriented	**In/Out Separation**	Power Crust [13]	Y	Y	N	Y	-	-
			Robust Cocone [65]	N	Y	N	Y	-	-
		Sculpting	Boissonnat's Sculpting [28]	N	N	N	N	-	$O(n^2 \log n)$
			Tight Cocone [64]	N	N	N	Y	-	-
			Convection [47]	Y	N	N	N	-	-
			Wrap [71, 70]	Y	N	N	N	-	-
			Peel [66]	Y	N	N	Y	-	-
		Graph Partitioning	Eigencrust [125]	N	Y	Y	N	-	$O(n\sqrt{n})$
			Graph Cuts Stereo [132, 99, 112]	Y	Y	Y	N	SP	-
			Graph Cuts Range [133]	N	Y	Y	N	SP	-

[1] N = Normals / LC = Local Connectivity / SP = Sensor Position.

[2] n refers to $|P|$.

Table 4.4: Surface reconstruction methods classification, approximation based. Columns refer to the implicit/explicit nature of the resulting surface, the ability of the method to deal with boundaries, noise, outliers, if it provides theoretical guaranties, if it uses additional information rather than the point cloud and its complexity overview, if provided by the authors.

			Properties						
		Methods	**Implicit/Explicit**	**Boundaries**	**Noise**	**Outliers**	**Guaranties**	**Additional Info.**[1]	**Complexity**[2]
Approximation Based	**Tangent Planes**	Hoppe's Method [104]	I	Y	N	N	N	-	$O(n \log n))$
		Natural Neighbors [29]	I	N	N	N	Y	-	-
		Markov Random Field [169]	I	N	Y	N	N	SP	-
		Voronoi-Based Variational [8]	I	N	Y	N	N	-	-
	Unsigned Distance	Unsigned Graph Cut [107]	I	N	Y	Y	N	N	-
		Signing the Unsigned [159, 90]	I	N	Y	Y	N	-	-
	Radial Basis Functions	Blobby [160]	I	N	N	N	N	N	-
		Fast RBF [43]	I	N	Y	N	N	N	-
		Multilevel Partition of Unity [165]	I	Y	Y	N	N	N	-
		MPU+RBF [166, 167]	I	Y	Y	N	N	N	-
		Smooth PU [161]	I	Y	Y	N	N	N	-
		Voronoi Centered RBF [181]	I	N	Y	N	N	-	-
		Cone Carving RBF [189]	I	N	Y	N	N	-	-
		Eigen RBF [206]	I	N	Y	N	N	-	-
		Support Vector Machines [186, 74]	I	N	Y	N	N	-	-
		Support Vector Regression [200]	I	N	Y	N	N	N	-
	Moving Least Squares	Point Set Surfaces [6, 2]	I	Y	Y	N	N	N	-
		Algebraic PSS [97]	I	Y	Y	N	N	-	-
		Implicit MLS [124]	I	Y	Y	N	Y	N	-
		Hermite MLS [4]	I	Y	Y	N	N	N	-
	Deformable Surfaces	Level Set [216]	I	N	N	N	N	-	$O(n + m)$
		Coulomb Potentials [111]	I	N	Y	Y	N	-	$O(o \log^2 o)$
		Touch-Expand Graph Cuts [135]	I	N	Y	N	N	-	-
		Discrete Membrane Shrinking [76]	I	N	N	N	N	-	-
		Self-Organizing Map [212]	E	N	N	N	N	-	-
		Growing Cell Structure [110, 109]	E	Y	N	N	N	-	-
		Implicit SOM [211]	I	N	Y	N	N	N	-
		Coarse-to-Fine Explicit [190]	E	N	N	N	N	-	-
	Gradient Enforcement	Fourier Transform [117]	I	N	Y	Y	N	N	$O(m^3 \log m + n)$
		Wavelets [153]	I	N	Y	Y	N	N	-
		Poisson [118, 119]	I	N	Y	Y	N	N	-
		Smooth Signed Distance [41]	I	N	Y	Y	N	N	-
	Integration	VRIP [60]	I	Y	Y	N	N	LC, SP	-
		Scale Space Fusion [83]	I	Y	Y	N	N	LC, SP	-
		Lifted Constrained Delaunay [180]	E	Y	N	N	N	SP	-
	L. Primitives	Spherical Cover [168]	E	Y	Y	N	N	-	-

[1] N = Normals / LC = Local Connectivity / SP = Sensor Position.

[2] n refers to $|P|$, m to the size of a regular grid, k to $K(p)$ and o to the maximum depth of an octree.

Chapter 5

Direct Point Set Surface Reconstruction

5.1 Introduction

After applying the pipeline described in Chapter 2, the scene is described in the form of a point set. However, due to the nature of the underwater medium, the retrieved point set is noisy and contains a large number of outliers.

After reviewing the state of the art in the previous chapter, we have identified the weaknesses of present methods when dealing with the surface reconstruction problem on raw corrupted point sets, possibly representing a bounded surface. Starting with this chapter, we propose four methods that try to solve our specific problematic of recovering the surface of an object from a set of points describing it while getting rid of outliers and noise and also trying to recover its boundaries. Furthermore, in order to provide a generic methodology, we do not rely on any additional information other than the input points themselves to reconstruct the surface.

In this chapter, we present a novel method for surface reconstruction able to provide a smooth surface approximation from a point set. This method can handle noise and cope with a large percentage of outliers, which, as mentioned above, is often the case with underwater datasets. Moreover, additional information such as per-point normals or local connectivity is not required. Despite the fact that this procedure focuses on applying this method to point sets coming from a computer vision pipeline, it is worth mentioning that its generality makes it applicable to all kinds of point set data. For instance, we report results of applying the method to a multibeam sonar dataset surveying an underwater 3D structure by registering different scans in a common frame, and then using this data as input for our method.

Our method was inspired by the RDT Delaunay refinement meshing paradigm. We propose answering the segment intersection queries required by the RDT mesher on-line by constructing a local surface in a neighborhood around the query segment and returning the intersection between both. The method takes into account possible aberrations in the data by using RANSAC in order to recover the local surface with the highest support. Furthermore, noise is not considered constant, and an automatic scale computation algorithm is used in order to guess the best RANSAC distance-to-model threshold to apply at each step. The main advantage of using the mesher algorithm as a base is to be able to tune the desired resolution of the resulting surface, a parametrization that is not available in most of the state of the art approaches. Besides, our method also provides the ability to recover surfaces with borders. This is desirable in an underwater scenario where normally a part of the seafloor is observed, and thus the retrieved point set does not represent a watertight surface.

When compared to other state-of-the-art methods, our proposal has the advantage of being able to process highly corrupted point sets without additional information. The loose requirements for the input and its low memory footprint make this method suitable for completing the modelling pipeline in the complex underwater environment.

5.2 Overview and Contributions

We base our method on the RDT Delaunay refinement meshing algorithm. This method only requires the user to provide an intersection detection between line segments and the surface in order to approximate the object with a mesh of triangles. Normally, a given approximation of the surface is devised, using, for example, some of the techniques described in Section 4.5, and then this approximation is meshed using RDT in order to get a surface whose triangles follow a given quality. However, what we aim to do is obtain an on-line answer to this kind of query, thus transforming a (re)meshing method into a surface reconstruction method. Given the query segment required by the mesher, a small, local part of the surface is computed using the points falling at a given distance from it. Then, this local surface is tested for intersection with the segment. Our local operator works directly on the raw, and possibly corrupted, point cloud, by using robust statistic techniques in order to avoid outliers when building the small surface patch. In this sense, we use a RANSAC method whose threshold is locally adaptive to the scale of the noise in the area. Thus, when compared to the state of the art, the main contribution of our method is the ability to deal with point sets corrupted with both variable noise and outliers without any kind of additional information.

Using the surface mesher as a base provides our method with the ability of specifying

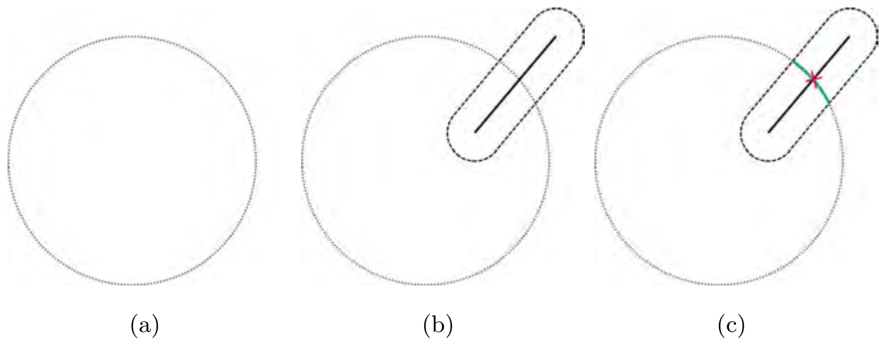


Figure 5.1: Schematic overview of the proposed on-line segment intersection query computation. (a) presents the input points, and (b) the query segment and its capsule neighborhood. As depicted in (c), a local surface is fitted to the points inside the capsule neighborhood and the intersection is computed.

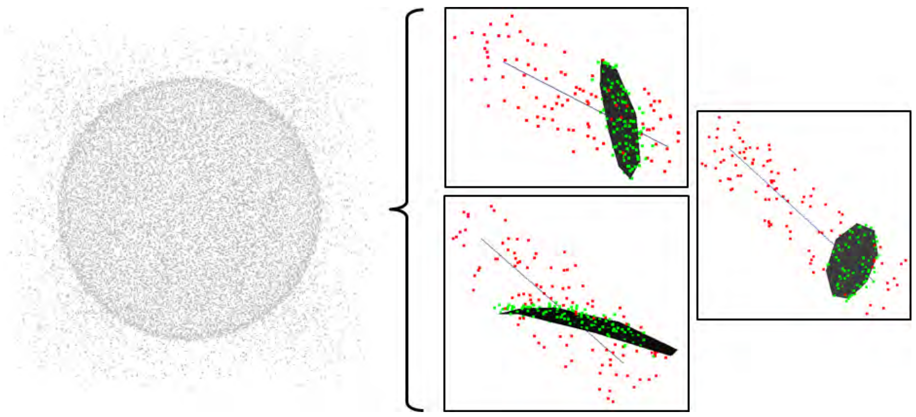


Figure 5.2: On the right are three examples of query segments required by the RDT algorithm to mesh the surface represented by the highly corrupted set of points on the left (a sphere), and the recovered local surfaces computed by our method used to solve the intersection computation. The results of our method applied to this point set can be found in Section 5.5.1.

the quality and properties of the resulting mesh in terms of shape and density of triangles. This allows having multiple resolutions of the same object by just tuning the parameters. Furthermore, it is worth noticing that the meshing algorithm works in a coarse to fine way, while our operator works with only a small part of the data at a time, in this way providing low memory requirements, which is the cornerstone for meshing large 3D maps.

Figure 5.1 shows a schematic idea of our method, and in Figure 5.2 one can find some examples of segment queries applied to a synthetic dataset. Note how even in the presence of outliers and noise, the local procedure is able to generate a correct local surface suited to answering the intersection query.

5.3 Online Intersection Computation

As already stated, the basis of our method is a surface meshing algorithm based on the concept of RDT and Delaunay refinement [30]. Recall from section 3.4.2 that the RDT refers to the subcomplex inside the 3D Delaunay triangulation formed by those triangles whose dual Voronoi edge intersect the surface the triangulation is restricted to. In our context, the most interesting property of this meshing approach is that it only requires devising an oracle that, given a Voronoi edge (i.e., a line segment query), computes its intersection with the inferred surface (if any).

This means that, as long as you have an approximation of the shape that you can query for line segment intersection detection, this representation can be meshed. In this chapter, our approach takes advantage of this property and answers the intersection queries locally by using local approximations of the surface built on demand using a small set of the input points at a time.

We use a local operator to answer the segment intersection queries required by the RDT mesher algorithm directly from the point set. Given the required query segment, we select the points that fall in a local neighborhood and compute a local low-degree surface with them. Then, the intersection between the local surface and the query segment is computed as the answer to the query.

Even if local, the mechanisms used ensure the correct generation of these surface patches when the data is corrupted with outliers and noise. Furthermore, the noise scale is not assumed to be fixed through the whole dataset, but locally adaptive to the area of interest at each local computation.

It is worth noting that, in order to compute the initial set of points for the RDT algorithm, random segments are generated inside the bounding sphere containing the points and apply the local intersection detection until we find a user defined number of initial points (20 in the presented results).

In the following section, our local operator is analyzed by defining the neighborhood around the query segment where the computations will be made. Then, in Section 5.3.2, the local surface we aim to extract inside the neighborhood is defined. Furthermore, Section 5.3.3 presents how we manage to compute this local surface in the presence of outliers, and Section 5.3.4 presents how we adapt to the measure of the noise for different parts of the dataset. Finally, the actual intersection between the segment and our local surface is described in Section 5.3.5.

5.3.1 Capsule Neighborhood

Let us name the input point set P. Given a query segment s, just the set of points near the segment are needed to compute the local intersection. This local neighborhood comprises all the points at a given distance from the query segment:

$$C(s) = \{p_i \in P | dist(p_i, s) < c\}, \tag{5.1}$$

where $dist(p_i, s)$ is the Euclidean distance, and c is the radial threshold describing the interest region. Thus, the points to take into account are those that fall inside a capsule (a.k.a. sweeping sphere or capped cylinder) of a given size around the segment. Note that this neighborhood may contain multiple structures as well as outliers. Throughout this section, we work with the case of a single $C(s)$. However, it is worth remembering that this computation is required several times inside the meshing algorithm.

5.3.2 Local Bivariate Quadric

Using the points in $C(s)$, we compute the intersection query required by the mesher algorithm. This is done by a Least-Squares fitting of a local surface patch using the points in this neighborhood. More precisely, we compute a Local Bivariate Quadric (LBQ). An LBQ is the quadratic approximation of a height function in a local reference frame:

$$f(x, y) = Ax^2 + By^2 + Cx + Dy + Exy + F. \tag{5.2}$$

Equation (5.2) can be computed from a minimum of 6 points using least-squares. In order to define the reference frame for the fitting, we observe that the Voronoi edge intersecting the surface defined by the RDT is close to being orthogonal to it. In fact, some authors have taken advantage of this observation in order to compute approximations of the normals of point sets using the Voronoi edges dual to their Delaunay triangulation (e.g., [10, 68, 8]). Based on this observation, it seems natural to fix the normal of the fitting plane to follow the direction of s. Thus, the fitting plane where the surface is computed has an orthonormal basis perpendicular to this direction. Finally, we construct the origin of the fitting frame to be also in the segment, by computing a centroid from all the points used for the fitting and project it orthogonally on s.

Furthermore, in order to rank the contribution of each point to the solution, they are given a weight proportional to their distance from s. As adopted in many other approaches, the weighting is defined by the well-known Gaussian function:

$$w(p_i) = w_i = e^{-\frac{dist(p_i,s)^2}{\sigma^2}}, \tag{5.3}$$

where σ is a parameter that we decided to fix at $\sigma = c$ in all our tests.

Having all these components defined, we aim to solve the following minimization problem:

$$\min \sum_{i=1}^{n} |w_i f(x_i, y_i) - w_i z_i|^2 , \tag{5.4}$$

which yields the following system of equations:

$$\sum_i^n w_i \begin{bmatrix} x_i^4 & x_i^2 y_i^2 & x_i^3 & x_i^2 y_i & x_i^3 y_i & x_i^2 \\ y_i^2 x_i^2 & y_i^4 & x_i y_i^2 & y_i^3 & x_i y_i^3 & y_i^2 \\ x_i^3 & x_i y_i^2 & x_i^2 & x_i y_i & x_i^2 y_i & x_i \\ x_i^2 y_i & y_i^3 & x_i y_i & y_i^2 & x_i y_i^2 & y_i \\ x_i^3 y_i & x_i y_i^3 & x_i^2 y_i & x_i y_i^2 & x_i^2 y_i^2 & x_i y_i \\ x_i^2 & y_i^2 & x_i & y_i & x_i y_i & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} = \sum_i^n w_i z_i \begin{bmatrix} x_i^2 \\ y_i^2 \\ x_i \\ y_i \\ x_i y_i \\ 1 \end{bmatrix} . \tag{5.5}$$

5.3.3 Outlier Rejection

In order to deal with outliers robustly, we do not use all the points in $C(s)$ to retrieve the LBQ. Instead, we use the LBQ as the model to compute inside a RANSAC procedure [79]. We chose the RANSAC method given its proven robustness in outlier rejection when used in the field of computer vision.

Each RANSAC iteration instantiates a candidate LBQ using 6 random points from $C(s)$ (the minimum number of points needed to compute the model). Then, the remaining points in $C(s)$ are tested for compatibility with the current model; that is, they are checked to fall at a distance smaller than a given threshold δ_d from the model. Each of the points falling inside this distance give a vote to this model, and after several iterations, the model that achieves the greatest support is the one selected. As a final step, the obtained model is refined by using least-squares with all the agreeing points. Note that since solving the Euclidean distance from a point to a quadric is a non-linear problem, we use the algebraic distance instead, which is faster to compute and proved to work well in the present case for all our scenarios. Another important problem to take into account is that, given a set of points, there will always be a candidate LBQ. Thus, we have to make sure that the selected LBQ has enough support to be considered correct. This is done through the δ_m parameter, which is the minimum number of points that have to agree with the computed model in order to trust this result.

Regarding the number of iterations needed to compute the model, we will use the probabilistic approach:

$$N = \log(1 - \varphi)/log(1 - (1 - \epsilon)^w), \tag{5.6}$$

where N is the number of iterations, φ is the desired probability for RANSAC to pick a sample of size w ($w = 6$ in the case of an LBQ) from a $C(s)$ free of outliers, and ϵ is the percentage of outliers inside $C(s)$ (see Hartley and Zisserman [103] for a detailed expansion of Equation (5.6)). We chose to fix $\varphi = 0.99$, and since ϵ is unknown, we initialize it with a worst-case estimate, and update it at each RANSAC iteration if the current computed model has greater support.

5.3.4 Locally Adjusted Noise Scale

When dealing with real datasets, it is not feasible to assume the noise scale at every part of the model to be constant. There are many factors that promote the non-uniformity of noise across P during its scanning. For the optical case, some examples of these factors might be the variable distance from the object at each frame, different conditions in illumination, visibility or turbidity, etc. Consequently, P presents variable noise measures for different parts of the dataset. We deal with this problem by using an automatic noise scale estimator.

As previously presented, the RANSAC method uses a distance threshold δ_d in order to decide, at each iteration, if a given point agrees with the current model under consideration. Obviously, δ_d is directly related to the amount of noise in a given $C(s)$: noisier parts require a bigger δ_d and viceversa. Thus, we aim to compute a scale which adapts to the needs of each specific neighborhood. Assuming the noise in the data to be Gaussian, the scale of this noise refers to its standard deviation σ. We use this measure of noise to automatically tune the RANSAC threshold as $\delta_d = 2.5\sigma$.

The model estimation problem is then enlarged: apart from seeking the parameters of our model, we need to compute its scale. The noise scale estimation is extracted from the residuals r_i of a point p_i. A residual is basically the difference between this point and its estimated value using the computed model. In our case, we do not know the model of our data, so it is a “chicken and egg problem”. In order to compute a scale estimation of our data, we will use the Modified Selective Statistical Estimator (MSSE) method [16]. This random sampling algorithm uses the Least K-th Squares (LKS) technique in order to obtain a reliable model, and then uses the residuals generated by this model to compute the scale of the noise.

Similar to the previously presented RANSAC algorithm, the LKS method generates at each iteration a candidate model Q using the minimum number of points possible. Then, using this model, it computes the residuals d_i for all points and sorts them. The idea is to keep track of the model with the minimum jth residual d_j, where j is selected as the minimum number of expected inliers. At the end of the LKS procedure, the model minimizing d_j is selected. In fact, the LKS method is a variant of the well-known LMedS method [176]. Selecting a $j = 0.5n$, n being the number of residuals, the LKS keeps track of the median, thus reproducing the LMedS method. We believe that the LKS method is preferable over the LMedS method because the latter has a breakdown point of 50%, and we do not want to assume at least half of our neighborhood being part of the inliers of our model. Note that more than a single model can be contained in our neighborhood, and thus, j has to represent the minimum number of points that are part of a structure inside the neighborhood. This minimum number of inliers is defined as the *quantile*, which we denote by o, and is a fraction of the k nearest neighbors. The number of iterations to complete is ruled again by Equation (5.6). Note however that the ϵ value has to be fixed in this case, as we cannot update it inside the method. We decided to use $f = 0.99$ and $\epsilon = 0.5$ as a balance between computational effort and robustness, having to generate $N = 293$ iterations to select a probably good model to compute the scale from.

Given a model computed by the LKS, its residuals can be computed and a scale measure can be extracted from them. There are many possible algorithms for solving the automatic scale estimation problem, two of the most well known are the Median and the Median Absolute Deviation (MAD) [177]. While simple to compute, these two methods again have a known breakdown point of 50%. This can be seen in the simple tests reproduced in Table 5.1. Since we want to be weaker when imposing the minimum number of points forming an LBQ inside our neighborhood, we keep following the idea proposed by MSSE, which consists of iteratively refining the scale measure. Starting from the j-th sorted residual corresponding to $j = ok$, the noise scale estimate is computed as follows:

$$\sigma_j = \frac{\sum_{i=1}^{j} r_i^2}{j - D}. \tag{5.7}$$

MSSE is based on the idea that incrementally taking into account more residuals (increasing j) generates σ_j measures that increase progressively, and a big jump will happen when the added residual comes from a point of a different model or an outlier:

$$r_j^2 > T^2 \sigma_j^2, \tag{5.8}$$

where T is a constant factor, which can be set to $T = 2.5$ if we assume Gaussian noise (98% of inliers will be identified as such). From Equations (5.7) and (5.8), the following inequation can be formulated:

$$\frac{\sigma_{j+1}^2}{\sigma_j^2} > \frac{T^2 - 1}{j - D + 1}. \tag{5.9}$$

Thus, the first outlier can be identified by checking the validity of this inequality, and the scale measure will be the σ_{j-1}; that is, the scale estimation previous to finding the big jump.

While it could be possible to compute the noise scale locally inside each $C(s)$, the computational effort it requires would make the method unfeasible. Consequently, a scale

Table 5.1: Noise scale computation example for points sampled on a plane, corrupted with noise and outliers. The number of input points is 1000, but a percentage of them (1st column) are generated following a unifom random distribution inside a slightly enlarged bounding box, and the inlier points are corrupted with Gaussian noise with varying amounts of standard deviation (2nd column). Using the LKS selection for the best model ($o = 0.2$), the scale is computed using MedianSE, MAD and MSSE. Due to the random nature of LKS, the procedure has been computed 100 times and the mean ($\hat{x}$) and standard deviation (σ) of the results are shown. Note how MSSE provides a close approximation of the original noise even in the case where the number of outliers is larger than 50%.

Data		**Scale Estimation**					
Outliers(%)	Noise (σ)	MedianSE		MAD		MSSE	
		$\hat{x}$	σ	$\hat{x}$	σ	$\hat{x}$	σ
0	0.03	0.03110	0.00126	0.01902	0.00108	0.02951	0.00113
20	0.01	0.01367	0.00051	0.00883	0.00052	0.01023	0.00049
40	0.05	0.07635	0.00333	0.05457	0.00230	0.05236	0.00247
75	0.03	0.06482	0.00275	0.05376	0.00224	0.03184	0.00172

value for each point is computed, previous to the meshing procedure, in order to alleviate the computational cost. We use a fixed k neighborhood around each point, and compute a scale measure using the procedure defined above. Then, during the meshing, from all the scale values corresponding to the points inside the current $C(s)$, a representative value is computed. In order to do this, we again apply MSSE, but to the set of scale measures associated to the points in $C(s)$. Consequently, this second MSSE step is not computed on residuals. Instead, we seek the first big jump in scales, which also corresponds to finding scales generated from another structure or from any outliers.

5.3.5 Local Bivariate Quadric - Segment Intersection

With the procedure presented so far, we are able to provide a local surface for each query segment. Thus, the last step needs to find the intersection with this LBQ surface. In order to compute the intersection between s and the LBQ, we cast the segment in its ray form and use its parametric equation:

$$s(d) = s_o + s_r d, \tag{5.10}$$

where $s_o = (x_o, y_o, z_o)$ is the starting point of the segment and $s_r = (x_r, y_r, z_r)$ its direction. Substituting Equation (5.10) in (5.2) results in:

$$\begin{aligned} &Ax_o^2 + Ax_r^2 d^2 + 2Ax_o x_r d + By_o^2 + By_r^2 d^2 + 2By_o y_r d + Cx_o + Cx_r d + Dy_o \\ &+ Dy_r d + Ex_o y_o + Ex_o y_r d + Ex_r y_o d + Ex_r y_r d^2 + f - z_0 - z_r d = 0. \end{aligned} \tag{5.11}$$

Taking into account the general quadratic equation:

$$\alpha d^2 + \beta d + \gamma = 0, \tag{5.12}$$

we can deduce its parameters from Equation (5.11):

$$\begin{aligned} \alpha &= Ax_r^2 + By_r^2 + Ex_r y_r \\ \beta &= 2Ax_o x_r + 2By_o y_r + Cx_r + Dy_r + Ex_o y_r + Ex_r y_o - z_r \\ \gamma &= ax_o^2 + By_o^2 + Cx_o + Dy_o + Ex_o y_o + F - z_o, \end{aligned} \tag{5.13}$$

and then obtain the two possible solutions for d using the general method. In order to ensure that the solution falls within the segment, we have to verify that $d < length(s)$.

The equations presented so far solve the general segment-LBQ intersection problem. Note however that we have fixed the normal of the fitting plane of the LBQ to coincide with the direction of the segment. Thus, the segment coincides with the fitting axis, and thus just a single intersection may occur. Having $x_o = y_o = x_r = y_r = 0$, the problem formulated in Equation (5.13) simplifies to:

$$d = -\frac{1}{F - z_o}. \tag{5.14}$$

The presented framework provides the intersections against correctly computed LBQs. However, during the computation of the LBQs, some problems may arise. The most obvious is that the structure in $C(s)$ might not be represented using an LBQ. This is the case for the example in Figure 5.3 (a), where the structure inside $C(s)$ could be too complex to be captured by an LBQ. Another problem emerges from the fact that RANSAC always returns an LBQ regardless of the number of points involved. As mentioned above, one of the parameters used to test the correctness of this surface is to impose a minimum number of points δ_i agreeing with it. However, there might be a case like the one depicted in Figure 5.3 (b), where $C(s)$ comprises some points, but clearly the surface they define is not supposed to intersect the segment since it is a hole. In order to deal with these cases, we only consider as valid the intersection points that are supported. This means that some inlier points must lay close to the intersection point in order to validate it. Thus, a user-defined minimum number of inlier points γ_i is required to fall inside a sphere centered on the intersection point and with a radius $\gamma_f c$ where $0 \geq \gamma_f \geq 1$, i.e., a fraction of c. Obviously, tunning γ_i and γ_f depends on the sparseness of the data compared to the desired c parameter. This simple heuristic also allows the method to return a null intersection in cases where the local surface represented by the points in $C(s)$ does not intersect the segment, but is close enough so that an LBQ intersecting the segment can be generated in $C(s)$. This is the problem shown in Figure 5.3 (c), and note that in this specific case, the most supported surface does not intersect the segment but there is

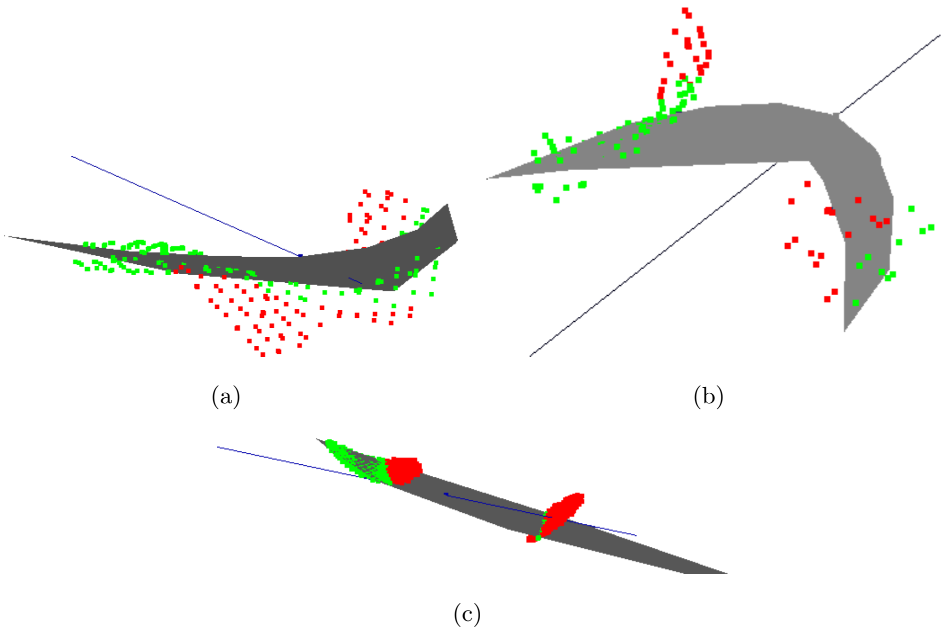
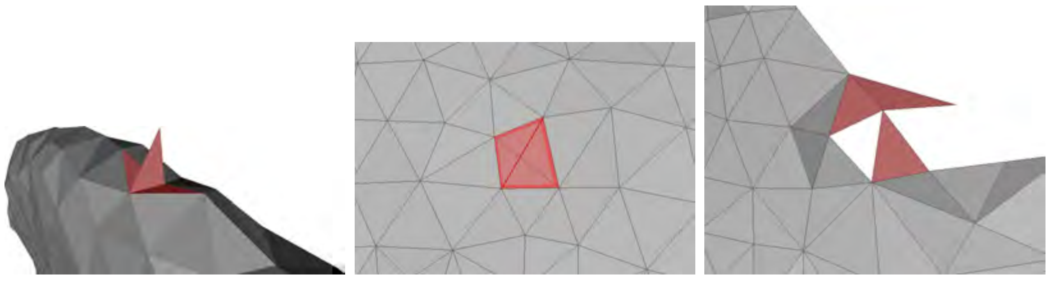


Figure 5.3: Problematic segment-LBQ intersection configurations: LBQ in grey, inliers in green and outliers in red. In (a), the part of the surface contained in $C(s)$ cannot be represented by an LBQ. In (b), the LBQ returned by RANSAC is filling a hole in the data, which is not desirable in most cases. Finally, in (c), the most supported LBQ is constructed from points falling far from the segment, and the computed LBQ is not correct. Note however that, on a second iteration, another LBQ will be generated from the set of outlier points, which in this case will clearly intersect the segment.

another cluster of points defining another local surface that intersects the segment. As presented above, we have to take into account that more than a single structure or LBQ may be contained inside the $C(s)$. For this reason, the model creation / intersection computation process is not unique: if the segment does not intersect the model, another model is computed from the set of remaining points (the outliers of the current model), and an intersection test is carried out again. The process stops when a valid intersection is detected, when no more models can be generated, or when a given number of trials is reached (3 in our implementation).

5.4 Post-processing

Although the proposed method provides satisfactory results in most cases, variability during intersection computation caused by noise may result in inconsistencies in the reconstructed mesh. It is obvious that we are not imposing any continuity on the surface by just performing local intersection computations. In this sense, the local surface computed



(a) Non-manifold edge (1) (b) Non-manifold edge (2) (c) Non-manifold vertex

Figure 5.4: Non-manifold structures, highlighted in red.

in a given part of the model is not continuous with a local surface computed nearby, even if most of the points taken into account are the same. Thus, the RDT mesher may not provide a manifold surface. For this reason, the non-manifold configurations that may arise in the mesh need to be filtered. In the following we present the filtering steps applied to solve these non-manifold cases:

- Non-manifold edges: Edges where more than two triangles meet generated by wrong intersection points detected near the surface. In most cases this aberration can be eliminated by removing the facets incident to the non-manifold edge whose three edges are not shared by another triangle (see Figure 5.4(a)). However, this test becomes invalid in the situation where the RDT includes a connected set of 4 facets, forming the boundary of a flat tetrahedron that shares four non-manifold edges with the reminder of the surface (as in Figure 5(b)). The special case is filtered out by selecting, out of these 4 facets, one of the two pair of facets incident along manifold edges.

- Non-manifold vertices: Vertices not surrounded by topological fans. In this case, since is difficult to tell locally which triangles are more likely to be eliminated, all the triangles incident to this vertex are removed.

Examples of these aberrations are illustrated in Figure 5.4. After filtering all the non-manifold configurations, some small sets of triangles may remain unconnected to the global surface. For this reason we remove the small connected components that are not likely to be part of the final surface.

Obviously, the presented manifoldness cleaning step will result in small holes in the surface that need to be filled. Furthermore, due to the large and variate possible cases of intersection between local surface and segment that can occur inside a given $C(s)$, some of the intersection queries may get a wrong result, as detailed in Section 5.3.5. This also produces some holes in the surface that need to be filled for aesthetic purposes. In order to do so, we use the method presented in [141]. This method provides the resulting filling to fairly interpolate the missing shape, using triangles with sizes similar to those on the border of the hole. The method applies 3 sequential steps to each hole. First, a minimum weight triangulation is devised in order to fill the hole linearly. Then, this initial hole-filling triangulation is refined following the apparent density and sizing of the border triangles of the hole. Finally, the triangulation is faired in a way so that the triangulation filling the hole smoothly approximates the shape based on the other parts of the surface surrounding the hole. The small holes to fill in our meshes are automatically detected as the set of border edges forming a closed loop, where this number of edges is smaller than a threshold. This threshold avoids filling large holes corresponding to undersampled parts or the borders of the recovered shape. Note that a procedure similar to the one proposed here (non-manifold configurations removal and hole filling) has shown to provide satisfactory results in Ohtake et al. [164].

In case of heavy noise, the surface retrieved by our method also presents some roughness. In order to alleviate these artifacts, we apply a Laplacian smoothing technique [101]. This method basically consists of moving each vertex on the mesh to the mean position defined by its neighbors. This smoothing procedure was not used on all the presented results, but only on a few of them (see Table 5.2) and, when applied, only a single iteration of Laplacian smoothing was enough to reduce the noise.

5.5 Results

We prove the validity of the presented pipeline on challenging real underwater datasets from different application areas. To better illustrate the excellent performance of the method, we briefly overview its behaviour when applied to standard range scan datasets. Then, we rigorously test the method on underwater datasets, focusing on the problems that optically generated 3D point clouds present in underwater environments and how they affect our method. In Table 5.2, one can find the parameters used in each execution. Note that there is a variability of values depending on the properties of each dataset. The more noise the dataset contains, the larger the number of k neighbors required for noise scale estimation. Moreover, for the intersection computation, the number of RANSAC inliers δ_i increases with larger noise and a greater quantity of outliers, while the number and localization of inliers near the detected intersection (governed by γ_f and γ_i parameters) becomes more restrictive. Besides, the radius c required for the capsule neighborhood is smaller for densely sampled point sets, and larger for sparser data. Finally, Table 5.3 shows the running times for each of the tested datasets.

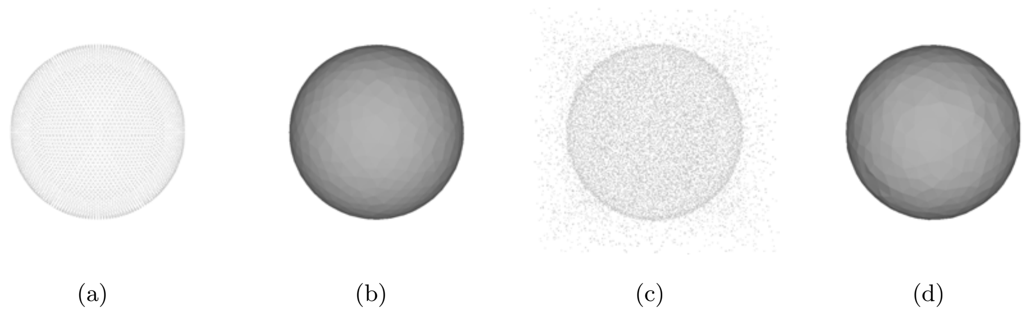


Figure 5.5: The Sphere dataset (10,242 points) shown in (a) is corrupted in (c) by adding a Gaussian displacement of $\sigma = 0.01$ (w.r.t. the bounding box diagonal) to all the points and adding 100% of randomly uniform distributed outliers inside its slightly enlarged bounding box. The surface for the clean version and the corrupted one are shown in (b) and (d) respectively.

5.5.1 Synthetic Dataset

Two small synthetic datasets are reported in this section to show the capabilities of the method under variable noise and outliers. The first dataset consists of a uniform sampling of a unit sphere. The results of our method under ideal conditions are illustrated in Figure 5.5 (a,b). In order to test the method with more realistic conditions, we decided to corrupt the original point set by adding a Gaussian noise with standard deviation $\sigma = 0.01BBD$ where BBD stands for the Bounding Box Diagonal of the input points. Additionally, we added 100% of outlier points by generating uniformly distributed points inside a slightly enlarged bounding box containing the original point set. Both the data points and the result of our method can be seen in Figure 5.5 (c,d), where the resulting surface has been faithfully recovered, despite not being as smooth as in the previous case.

The second synthetic dataset has been generated by sampling a torus, but this time non-uniformly sampled. The point set and its surface retrieved by our method are shown in Figure 5.6 (a,b). In order to test the behaviour of our method, not only under non-uniform sampling but also under non-uninform noise, we added noise to this dataset that varies from left to right in Figure 5.7 (a), from $\sigma = 0BBD$ to $\sigma = 0.005BBD$. The behaviour of our method can be seen in Figure 5.7 (b), where one can appreciate that the surface is hole ridden and quite bumpy. In this situation, the hole filling procedure, followed by the Laplacian smoothing, allows the method to recover a more pleasant representation of the torus. The results of these two steps can be seen in Figure 5.7 (c) and (d) respectively.

5.5.2 Range Scans

Range scanning technology has become a tool of great importance in land robotics systems. High-resolution indoor object reconstruction has also benefitted from the new achieve-

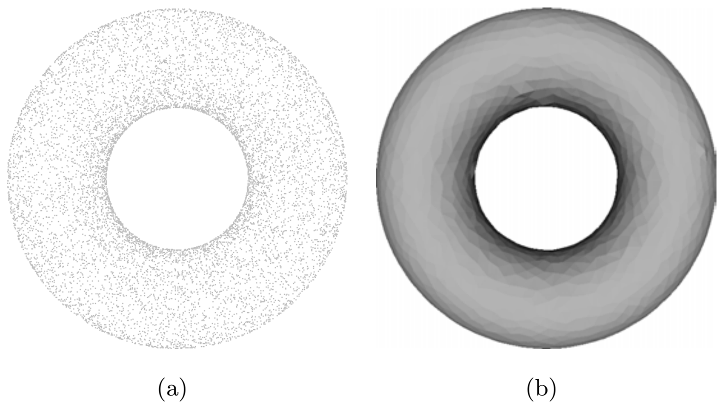


Figure 5.6: Using the noise free version (a) of the Torus dataset (10,000 points), the surface on (b) is generated using our method.

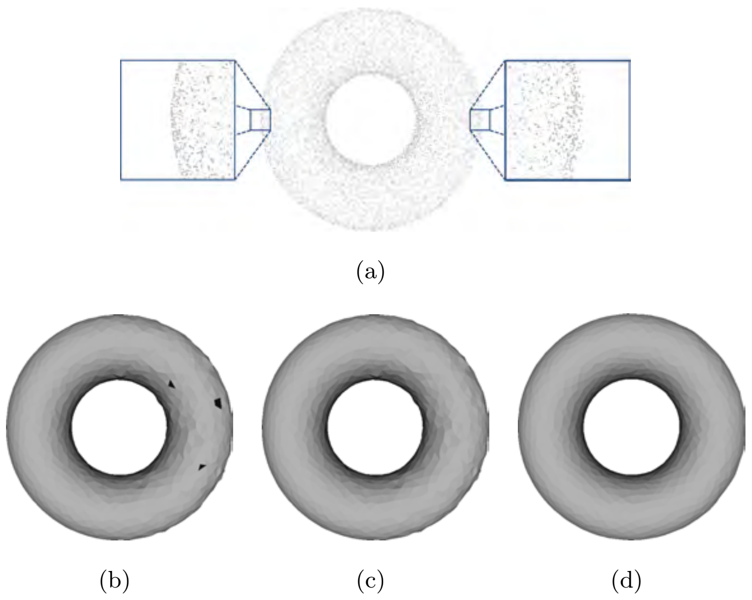


Figure 5.7: The noise corrupted version of the Torus dataset (a), causes the surface recovered by our method (b) to be bumpy and defected with holes. The hole filling (c) and smoothing (d) procedures allow the recovery of a more faithful approximation of the original object.

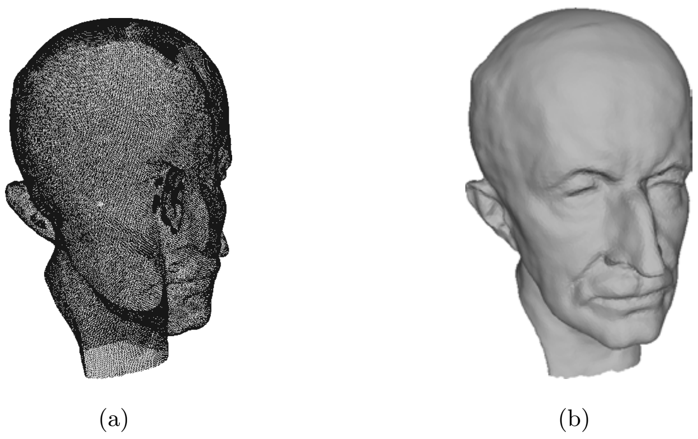


Figure 5.8: Results of our method applied to the Max Planck point set (199,169 points).

ments in range finders. We illustrate the behaviour of our system when applied to some of the benchmark datasets from this kind of sensor. Let us start with the Max Planck model, a close-to-ideal dataset which is regularly sampled and not corrupted by noise. We can see in Figure 5.8 how in this case the reconstructed surface faithfully represents the object. It is worth noticing that when no noise is present in the data the smoothing step is not required.

On the other hand, the effect of the cleaning and hole filling post-processing methods is shown in Figure 5.9. Figure 5.9 (b) presents the raw surface as extracted by our method, where some holes (black spots) are visible. After the non-manifold configurations cleaning step, Figure 5.9 (c), even more holes appear. After the hole filling, the final surface is presented in Figure 5.9 (d).

As previously shown, since the quality of the resulting mesh is user-defined, our method is able to create multiple resolution versions of the same object. Figure 5.10 shows an example of this property applied to the Elephant dataset.

Finally, we apply our method to a noisy dataset, namely the Stanford Bunny, where measurement variability and registration errors are present. In this case, and due to the lack of continuity enforcement on the surface retrieved by our method, the result obtained shows a bumpy surface in the areas where the mentioned errors are more emphasized (Figure 5.11 (b)). Here, as seen in the Torus point set, the optional smoothing step can be applied to correct the results.

In order to demonstrate the resilience to outliers of our method, and given that these range scans do not usually contain many outliers, we corrupted the Stanford Bunny dataset by adding 100% of outliers. These outliers have been added by generating a set of random points uniformly distributed inside the bounding box of the original dataset slightly enlarged by 5%. The results can be seen in Figure 5.12, where one can see that the method

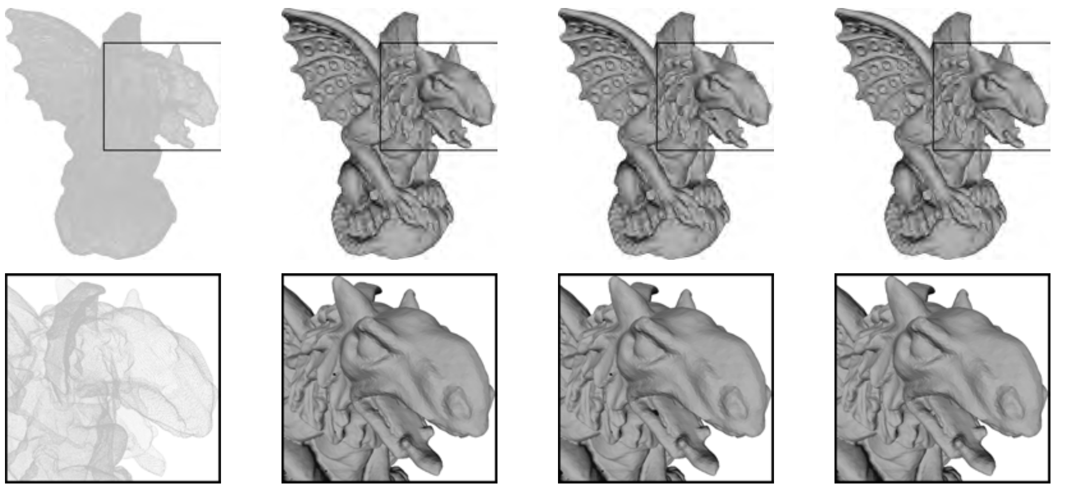

(a) Point set (b) Surface (c) Cleaned surface (d) Hole filled surface

Figure 5.9: The application of our method to the Gargoyle point set (863,210 points) shown in (a) raises the surface in (b), where holes are directly visible as black spots. After the non-manifoldness cleaning step in (c), even more holes appear. By using the hole filling procedure, these small imperfections are corrected, as can be observed in (d).

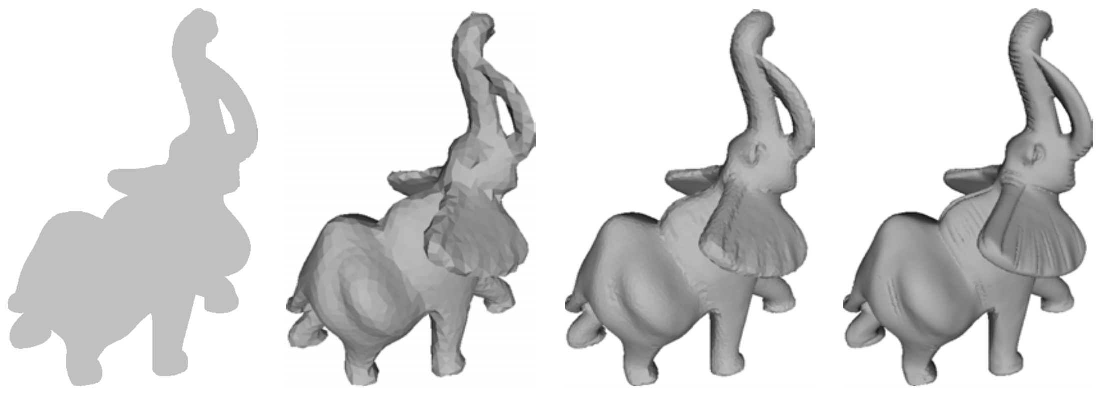

(a) (b) (c) (d)

Figure 5.10: The Elephant dataset (1,537,283 points), shown in (a), is used as input for our method to create multiple surfaces, shown from (b) to (d), at increasing resolution.

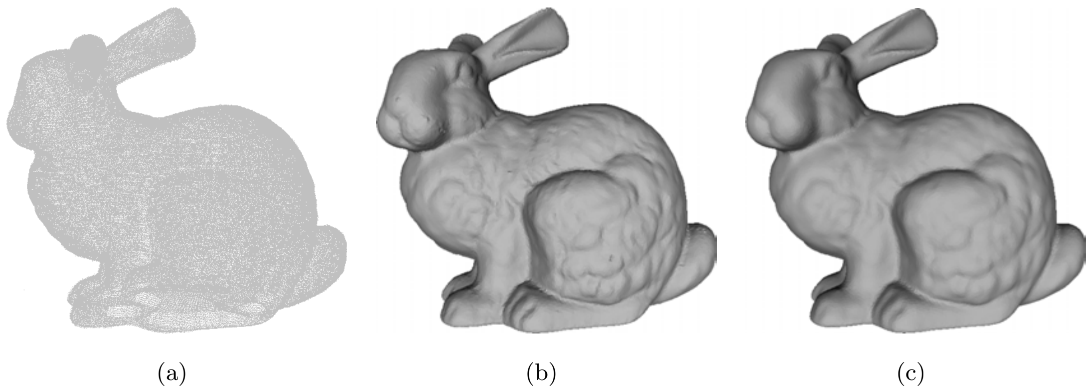


Figure 5.11: In (a), the Stanford Bunny dataset (362,272 points) is presented. Note in (b) how the recovered surface is quite bumpy near noisier areas. In (c), one can see the usefulness of the smoothing step in these cases.

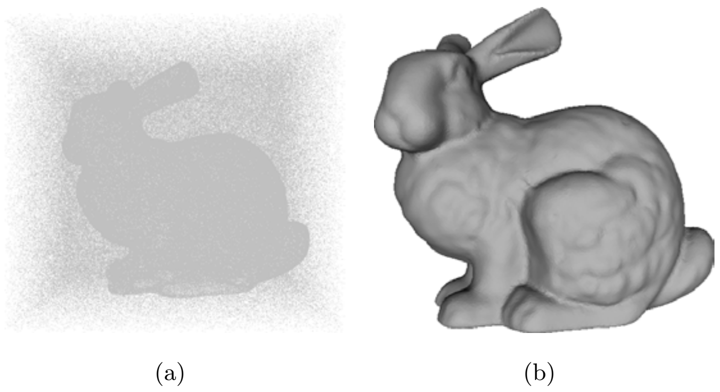


Figure 5.12: (a) shows the Stanford Bunny dataset, corrupted with 100% of uniform randomly distributed outliers (724,544 points), while (b) presents the resulting surface as retrieved by our method.

is still able to provide a correct reconstruction. Note how even in the presence of this vast amount of outliers, the result provided by our method is quite similar to those ones presented in Figure 5.11 (b) and (c).

In underwater robotics, range sensors come in the form of multibeam sonar. In order to demonstrate the versatility of the presented method, we show the behaviour of the method when applied to multibeam sonar scans. Contrary to most multibeam surveys, it is not that obvious in this case whether there is a plane proxy where this raw point set could be approximated using a 2.5D heightmap (check the structure in Figure 5.13, left). Thus, a method working on unconstrained 3D like the one presented might be more desirable in order to reconstruct a surface mesh from these points. The applicability of our method is demonstrated in the right part of Figure 5.13, where the recovered surface

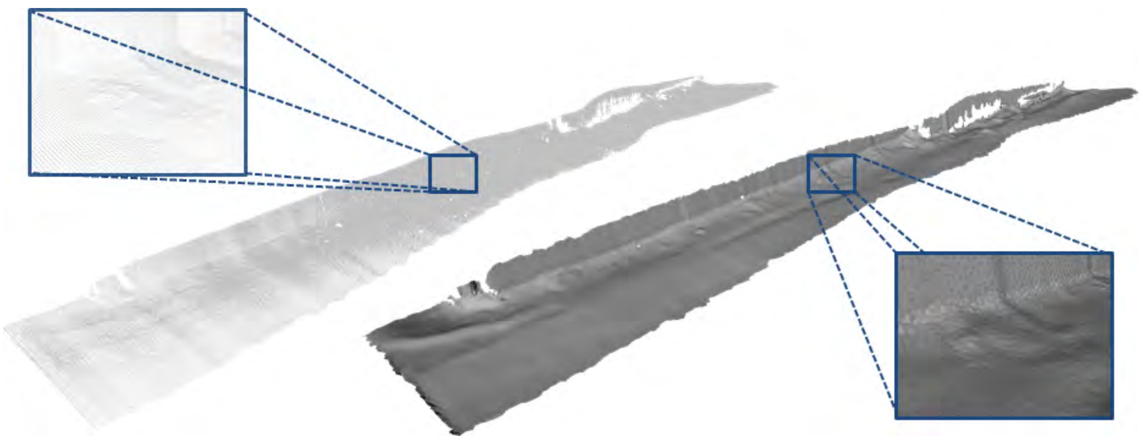


Figure 5.13: Multibeam sonar reconstruction (413,873 points). Instead of the standard downward-looking configuration, the transponder was installed laterally, giving rise to a full 3D dataset.

is presented. This application sets the path for the development of sonar scanning of underwater structures in more general configurations than the typical downward/forward viewing with respect to the scene.

5.5.3 Underwater Multi-view Stereo Datasets

Finally, in this section we present the results of our method when applied to optical multi-view seafloor reconstruction. The optical reconstruction technique used to retrieve the point set used as input for our method is a common multi-view plane sweeping technique similar to the one in Yang and Pollefeys [209]. This kind of technique provides a dense representation of the scene, as each image contributes to the point set with a dense depth map. However, the outliers of each depth map accumulate for each view. Furthermore, reconstructing each depth map separately for each view does not enforce coherence between them, which translates into double contours when registration errors are present. Finally, in order to take advantage of the richness provided by the texture in the images, we also present for each dataset its texture-mapped version. For each triangle in the model, we use the texture corresponding to the view that maximizes the projected area in image space. This simple technique has proven to be useful in the underwater media [87], as it offers a good tradeoff between the orthogonality of the view direction with respect to the orientation of each triangle and promotes views taken closer to the scene, i.e., less affected by the absorption and scattering of the medium.

Consider the behaviour of our method with this type of dataset, starting with a simple dataset acquired in shallow waters. This shallow-water dataset presents an almost ideal scenario, as the common challenges appearing in underwater imaging are not present. As can be seen in Figure 5.14, our method is able to extract a smooth surface under these

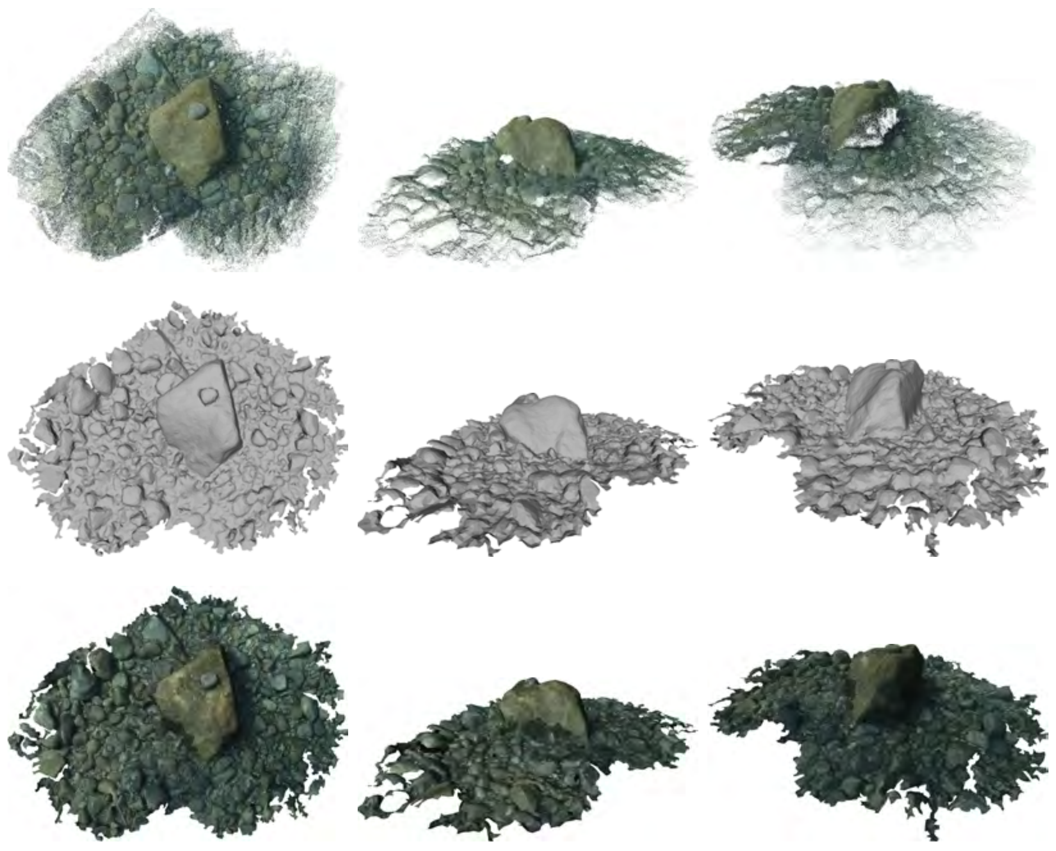


Figure 5.14: Shallow Water dataset reconstruction. From first to last row: point set (1,856,271 points), meshed surface, and textured surface.

conditions, preserving the fine details of the scene.

A more complicated dataset, also covering a larger area, is presented in Figure 5.15. This dataset consists of a survey of a coral reef, acquired at a relatively low depth. Even if the illumination conditions are not a problem, the retrieved point set presents a clear example of non-uniform sampling. This poses a challenge to our algorithm, as no sampling means more holes in our surface. In this case, we decided not to fill them in the final surface, as the hole is not closed and would require filling the entire border where the sampling of the surface ends. Taking a closer look at the places where the sampling is unsatisfactory, one can see that they correspond to vertical walls. These vertical walls were only shallowly observed during the survey, since a downward-looking camera configuration was used. This resulted in fewer samples covering those areas, promoting the dramatic change in the sampling rate between the top of these mound-like shapes and their lateral parts.

In the cases shown so far, the survey of the scene has been carried out with a downward-looking configuration. This does not demand the application of our method, as an ap-

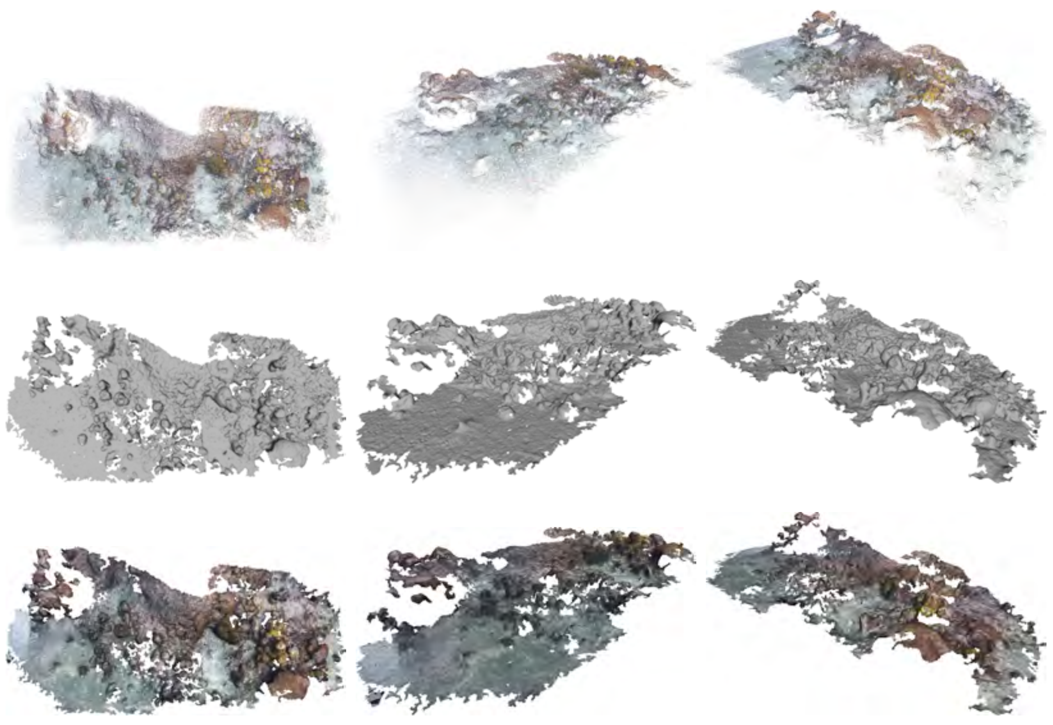


Figure 5.15: Coral Reef dataset reconstruction. From first to last row: point set (1,656,413 points), meshed surface, and textured surface.

proximation in the form of a height map could also be provided, preserving more or less the overall shape of the scene. However, the case presented in Figure 5.18 presents a more general camera configuration for the survey of an underwater hydrothermal vent. This hydrothermal vent, called the *Tour Eiffel*, has been the target of many geological, chemical and biological studies in the last decade [134]. It is located on the Mid-Atlantic Ridge, at around 1700m depth, and is more than 15m high. Obviously, the details of this intrincate shape cannot be recovered using the common downward-looking configuration. In this case, the survey was carried out using a forward-looking camera configuration, with the robot going around the object several times (see the camera trajectory in Figure 2.13). Note in Figure 5.16, how the input sequence is more challenging than in the other cases, as the attenuation of the light with distance is strong, and the high depths at which it was collected forced the use of artificial lightning, causing non-uniform illumination across the images and emphasizing the blurring and scattering produced by suspended particles in the medium. Due to these phenomenons, the point set retrieved is noisier and presents several outliers, as depicted in more detail in Figure 5.17. However, in Figure 5.18 we can see that even in this case our method obtains a correct surface following the main shape, at the same time disregarding the outliers. In order to validate the output of our method, we compared our results with those of some of the methods in the state of the

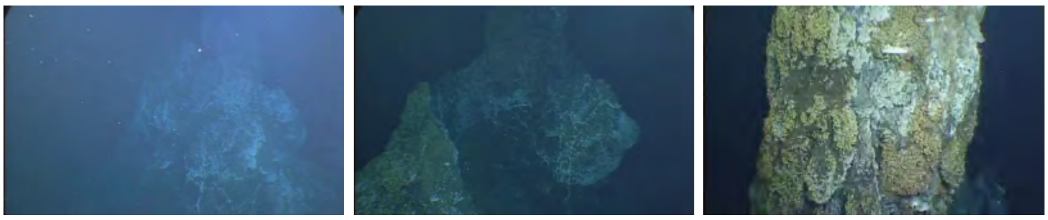


Figure 5.16: Three sample images of the Tour Eiffel sequence (from a total of 928). This challenging sequence presents common problems in underwater imaging: blurring, attenuation of light with distance and scattering.

art. The first row in Figure 5.19 shows the resulting surfaces after applying the most relevant methods. The selected methods were MPU [165], Robust Cocone [65], APSS [97] (its MeshLab implementation [53]), and Poisson [118]. Given the global view of these methods, they are able to deal with outliers to some extent. However, the vast number of outlier points posed by this dataset doomed them to failure. The result obtained by MPU, illustrated in Figure 5.19 (a), shows a spiky surface that does not resemble the object visible in the original image sequence, and also some hallucinated off-surfaces created due to outliers. In Figure 5.19 (b), we can see that the surface retrieved by the Robust Cocone method encloses most of the points (including the outliers), giving rise to a hull whose triangles do not pass near the denser parts of the original point set, where the surface is actually described. All the same, the results of the APSS method in 5.19 (c) are similar to the MPU case, presenting again a bumpy mesh that does not resemble the expected surface. Finally, in Figure 5.19 (d), one can find the results for the Poisson method [118], a widely used and cited method nowadays. Compared to the other methods, it achieves a better reconstruction in the sense that most of the object's shape is correctly recovered as a smooth surface. However, it also shows a non-existent off-surface at the back of the model that should be filtered manually. It is worth noting that, except for the Robust Cocone method, all the presented methods require oriented point sets, i.e., a normal for each point must be known. For all the cases presented, normals have been computed using the method of Hoppe et al. [104] using a $k = 100$, which we found to be a good tradeoff given the number of points in the dataset and its noise. On the contrary, our method works directly with the raw point sets.

We present next an even noisier dataset corresponding to a survey of a shipwreck of the XVIIth century located near the coast of Toulon (France) at around 70m depth. The ship, called *La Lune*, belonged to the fleet of King Louis XIV, and the good conditions of its remainings presents a rich and important archeological source of information. This dataset is part of a detailed survey around the cannons that are still preserved on the ocean floor. The data represents a cannon, with two cauldrons nearby. The survey followed a

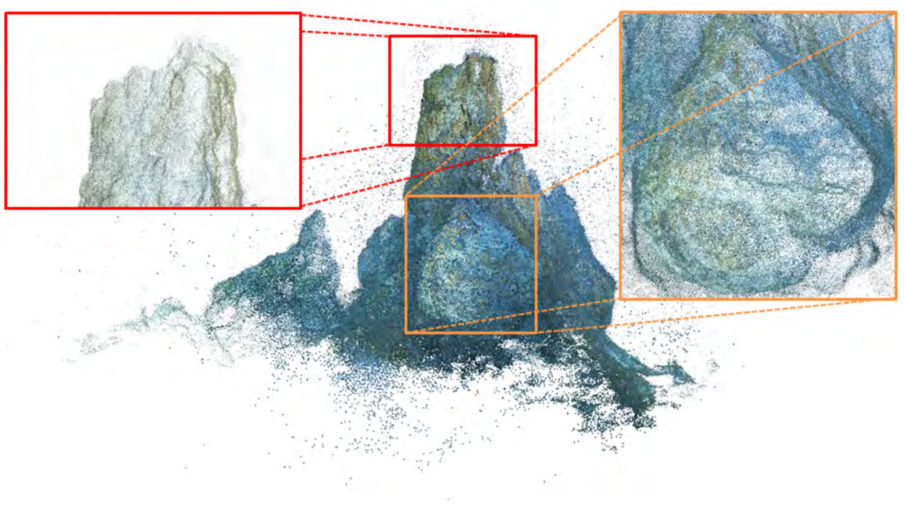


Figure 5.17: Close-up on the point set of the Tour Eiffel dataset (1,368,115 points). Note the high number of outliers.

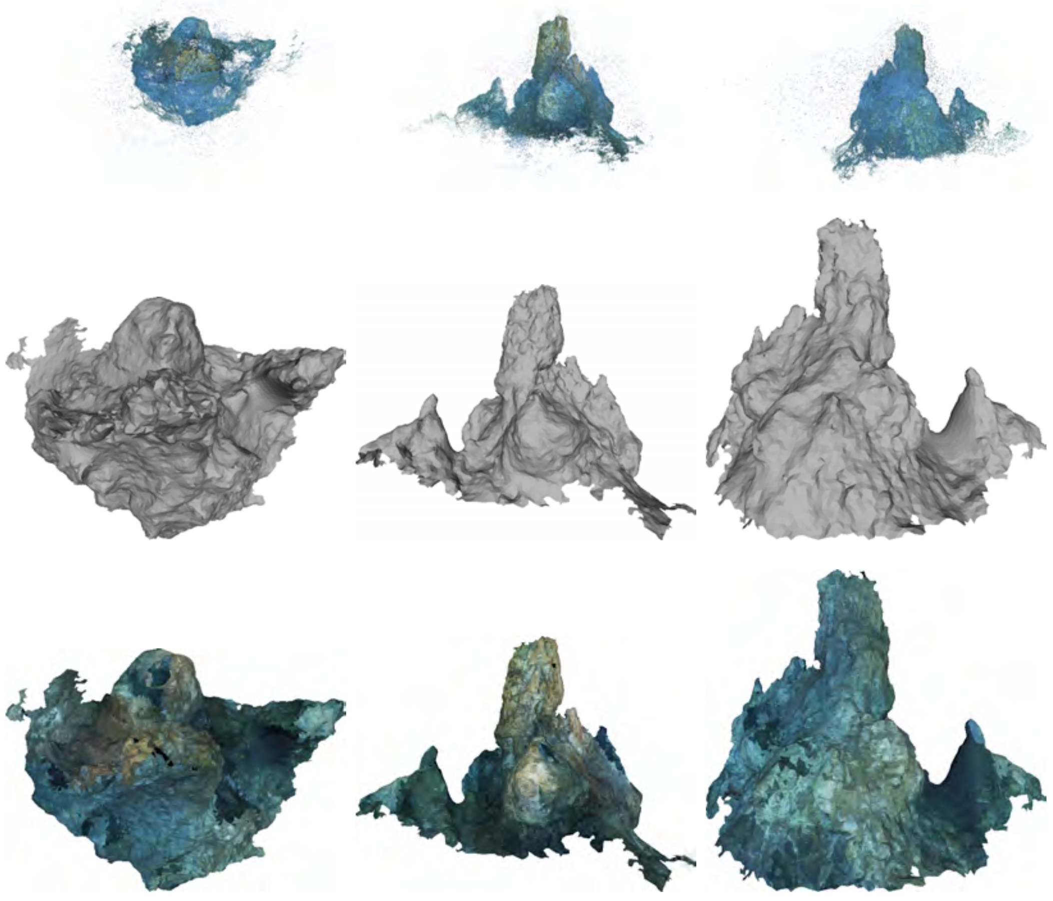
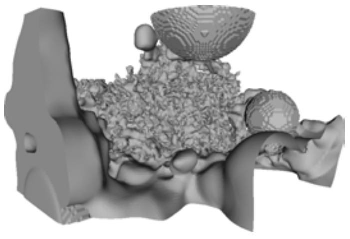
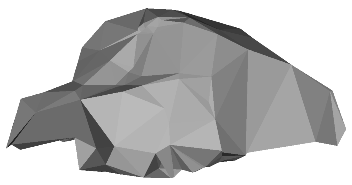
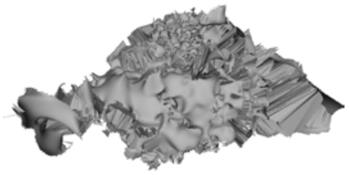


Figure 5.18: The Tour Eiffel dataset, reconstruction of an underwater hydrothermal vent. From first to last row: point set, meshed surface, and textured surface. Note how the surface is correctly reconstructed even in presence of a high number of outliers.


(a) MPU [165]


(b) Robust Cocone [65]


(c) APSS [97]

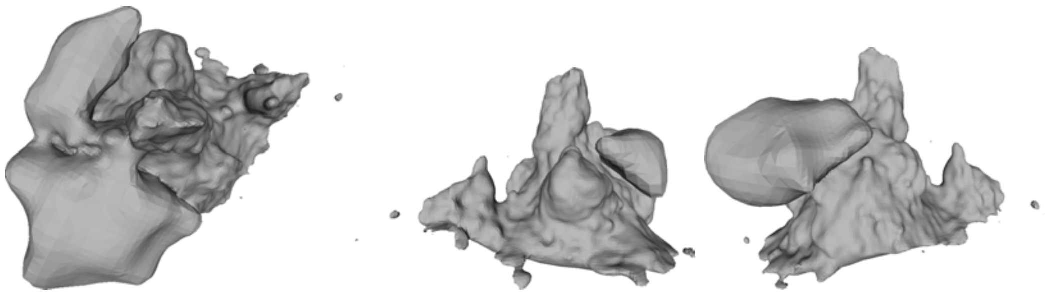

(d) Poisson, 3 views [118]

Figure 5.19: Results of some of the state-of-the-art surface reconstruction methods applied to the Tour Eiffel dataset. With the exception of Robust Cocone, all the methods tested require per-point normals, while our method does not. In all cases, we compute them using the method of [104] with a $k = 100$. Note how the original defect-ladden point set prevents the reconstruction of a correct surface in (a)(b) and (c), while producing artifacts in (d).

configuration similar to the one presented before, with a slightly slanted forward-looking camera configuration, and performing several turns over the object of interest. However, the conditions of the data acquisition were even worse than in the Tour Eiffel case in terms of scattering and visibility. In this case, the imaging data was extracted from an interlaced PAL camera capturing at grey scale. Some of the images in the sequence can be seen in Figure 5.20. The problems depicted resulted in a bad estimation of the camera positions during the structure-from-motion process. The error during the registration of the cameras propagates in subsequent steps, giving rise to a point set with a lot of outliers and registration errors between depth maps. These problems can be seen in more detail in Figure 5.21. Note that while our method has been designed to deal with outliers and variable noise across the same point set, registration errors are not solved using our pipeline. Thus, *double contours* rising from incorrect registration of the cameras are likely to be reconstructed separately. This means that the surface we are trying to mesh is inherently non-manifold. Thus, the surface mesher also returns a non-manifold surface in those parts where the registration errors are more pronounced. These problems are alleviated if the required resolution for the model is decreased. The results obtained show an incomplete model with holes and low resolution. Despite this fact, our method obtains the overall shape of the model, with holes in large undersampled areas and recovering the borders of the surface. The overall shape of the cannon is clearly visible and note how the open top of the cauldron seen in the second image of Figure 5.20 has been correctly reconstructed. The quality and resolution of our result is not as fine as one would desire, but the dataset is complicated enough to also pose problems to the most relevant methods in the state of the art. In Figure 5.23 one can see the results of the Poisson method [118], is also having problems reconstructing this scene. As previously mentioned, this method relies on knowing per-point normals, which had to be computed in our case following the method of [104]. The critical parameter of Hoppe's method depends on the k-neighborhood required and the correctness of these normals will influence the results of the Poisson method. The first row in Figure 5.23 shows the results of applying an increasing k for the same octree depth for Poisson, which returns a surface similar in resolution to ours. Even in the case of using a $k = 50$ neighborhood, note the nonexistent off-surfaces created by the outliers. Also, the top of the cauldron is closed, as this method seeks a water-tight surface. In the case of using a higher depth for the octree, as in the second row in Figure 5.23, the surface is more detailed, but the surface is also highly extrapolated in parts where not enough samples are taken (e.g., under the cannon) or in parts where borders should be present (the top of the cauldron or the limits of the area surveyed).

Finally, the last dataset comes again from an archeological survey, in this case including two pottery elements. Obviously, the conditions during image capture were far better than

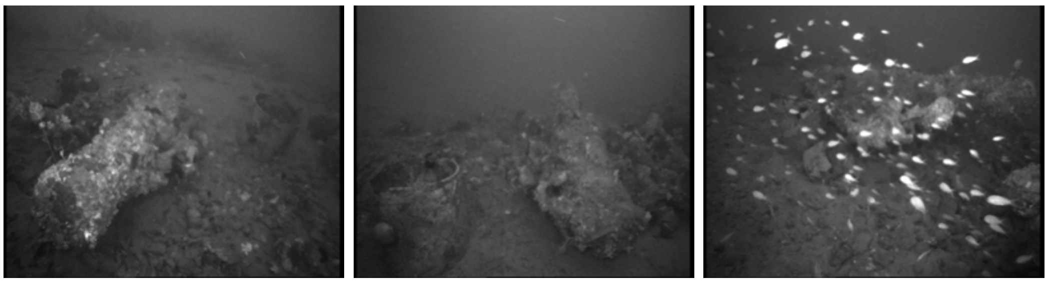


Figure 5.20: Three sample images of the La Lune sequence (from a total of 981). Note how the blurriness of the scene, the moving objects, light attenuation with distance, and non-uniform lightning make the processing of this sequence difficult.

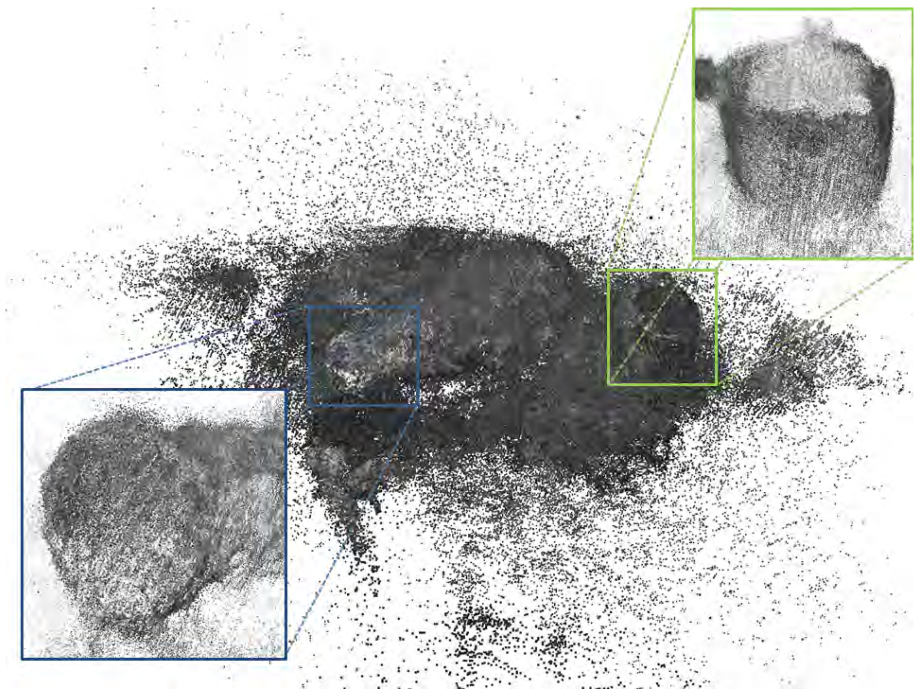


Figure 5.21: Close-up of the noisy point set of the La Lune dataset (1,137,820 points).

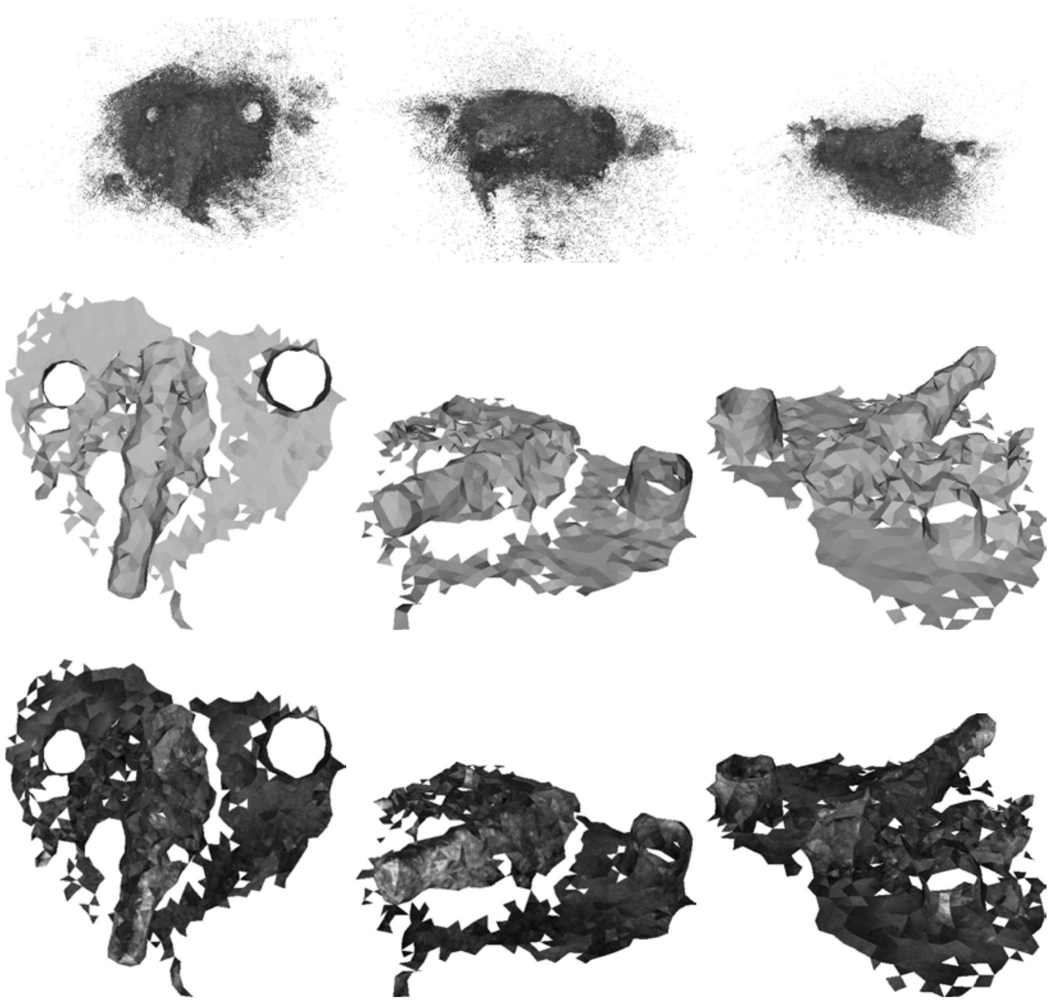
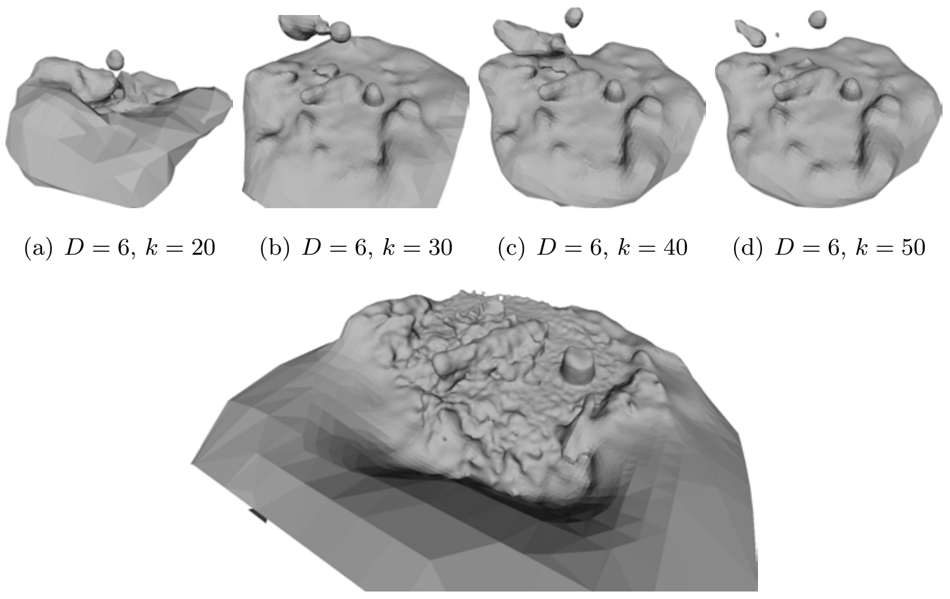


Figure 5.22: La Lune dataset reconstruction, a cannon and cauldrons on a shipwreck. From first to last row: point set, meshed surface, and textured surface.



(a) $D = 6$, $k = 20$ (b) $D = 6$, $k = 30$ (c) $D = 6$, $k = 40$ (d) $D = 6$, $k = 50$

(e) $D = 8$, $k = 50$

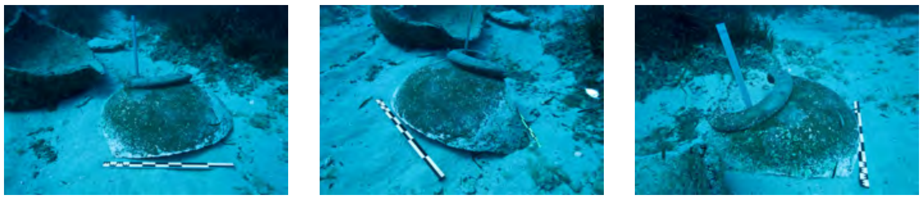
Figure 5.23: La Lune dataset, results using the Poisson method [118]. In each figure, D stands for the octree depth required for Poisson, and k for the number of neighbors to take into account during Hoppe's normal computation method. Note in the top row how different k values during normal inferring modify the output of the Poisson method.

in the previous case, leading to clear images of high quality. These images allowed us to recover a less noisier point cloud. Using this point set we are able to retrieve a smooth point cloud, which is suitable for surface reconstruction using our method. Figure 5.24 shows these results and demonstrates how, under less noisier conditions, our method retrieves smooth surfaces that faithfully resemble the observed object.

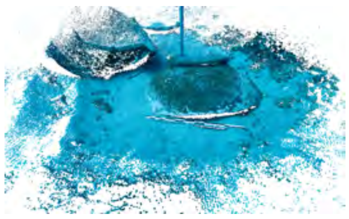
5.6 Conclusions and Future Work

We have presented a surface reconstruction method applied to raw point sets able to provide smooth approximations of surfaces in the form of triangle meshes. Its applicability to fields such as biology, geology and archeology has been demonstrated, providing a detailed representation of an area of interest. We tested the results against those of the state of the art, proving that it can provide reconstructions of similar quality while presenting the advantage of being able to deal with outliers and noise in the data, common situations in underwater datasets.

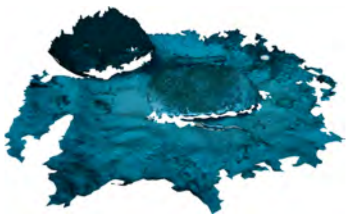
The applicability of our method has been demonstrated against a variety of datasets presenting different characteristics, and including synthetic data, range scans and optical reconstructions. In all cases, the method achieves a faithful reconstruction of the surface



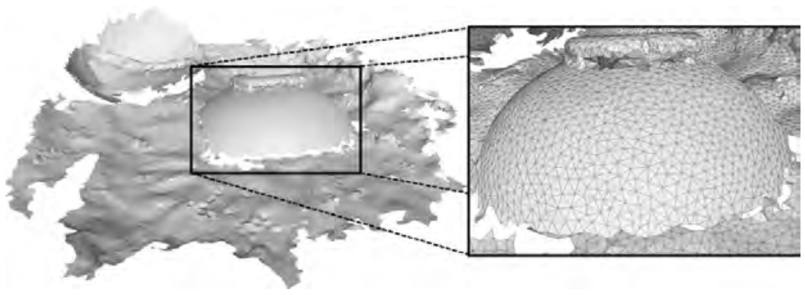
(a) Sample Images



(b) Point Set



(c) Textured Surface



(d) Surface

Figure 5.24: Pottery dataset. The first row (a) shows some input images (from a total of 39). We present the input point set (1,278,499 points) in (b), the reconstructed surface in (d) and its texture mapped version in (c). Note how, given less noisier conditions, the retrieved surface is smooth and faithfully represents the geometry of the observed object.

Table 5.2: Table showing the parameters used to generate the results presented in Section 5.5. Footnotes clarify the meaning of each column.

Datasets			**Noise**		**Intersection**				**Meshing**	
Name	Num. Points	Figure	k*	o†	δ_i ‡	c§	γ_f	γ_i	α_r/α_d¶	S‖
Sphere	10,242	5.5(b)	500	0.3	6	0.1	0.5	3	0.1	N
Corrupted Sphere	20,484	5.5(d)	500	0.3	75	0.1	0.5	3	0.1	N
Torus	10,000	5.6(b)	500	0.3	6	0.1	0.5	3	1	N
Corrupted Torus	10,000	5.7(d)	500	0.3	15	0.1	0.5	5	1	Y
Max Planck	199,169	5.8(b)	500	0.5	6	0.02	0.5	3	2.25	N
Gargoyle	863,210	5.9(d)	500	0.5	6	0.01	0.5	2	0.5	N
Elephant	1,537,283	5.10(d)	200	0.3	6	0.01	0.5	2	1	N
Stanford Bunny	362,272	5.11(c)	500	0.3	20	0.02	0.5	3	0.001	Y
Corrupted Stanford Bunny	724,544	5.12(b)	1000	0.3	30	0.02	0.5	3	0.001	Y
Multibeam	413,873	5.13	500	0.3	6	0.01	0.5	3	0.5	N
Shallow Water scene	1,856,271	5.14	2000	0.3	50	0.015	0.3	5	0.1	N
Coral Reef	1,656,413	5.15	2000	0.3	50	0.01	0.3	5	0.1	N
Tour Eiffel	1,368,115	5.18	2000	0.3	50	0.01	0.5	10	0.07	N
La Lune	1,137,820	5.22	2000	0.3	50	0.02	0.3	15	0.5	N
Pottery	1,278,499	5.24	2000	0.3	50	0.015	0.3	5	0.1	N

*Number of nearest neighbors taken into account.

†Quantile.

‡Minimum number of inliers for RANSAC.

§Radius to take into account when generating $C(s)$, w.r.t. Bounding Sphere's Radius.

¶Meshing parameters, fixed to be the same to achieve a more isotropic result. We omit α_a, as we fix it to $\alpha_a = 0$.

‖Smoothing performed? (Y = yes / N = no).

Table 5.3: Table showing the running times for the two main steps in the algorithm. All the experiments have been performed on a machine with an Intel i7-3770 CPU at 3.4 GHz with 32 GB of RAM.

Datasets		**Timings (s)**	
Name	Figure	Scale	Meshing
Sphere	5.5(b)	248.57	12.11
Corrupted Sphere	5.5(d)	511.84	288.60
Torus	5.6(b)	250.36	81.76
Corrupted Torus	5.7(d)	243.09	67.54
Max Planck	5.8(b)	4872.52	573.30
Gargoyle	5.9(d)	21842.40	4652.49
Elephant	5.10(d)	27450.20	4616.36
Stanford Bunny	5.11(c)	8544.38	78.95
Corrupted Stanford Bunny	5.12(b)	25941.20	7253.89
Multibeam	5.13	10496.10	6639.78
Shallow Water scene	5.14	120014.00	3486.59
Coral Reef	5.15	107785.00	6804.23
Tour Eiffel	5.18	89091.20	1817.42
La Lune	5.22	74888.50	1497.01
Pottery	5.24	83453.90	36269.40

of the object even in the presence of high levels of noise and outliers, without requiring any pre-processing step to alleviate these aberrations, nor any additional information than the position of the points themselves. Of special interest are the datasets of Tour Eiffel and La Lune, where the poor quality of the imagery used to construct the point cloud led to massively corrupted data. Nonetheless, our method has proven to handle such data and retrieve surfaces reconstructing the surveyed objects, outperforming the methods on the state of the art.

Despite the fact that the method presented has proven to work well in a wide range of scenarios, it can still be improved. On one hand, its computational complexity is higher than that of its counterparts. On the other hand, the method uses a fixed radial neighborhood size. While this is enough for the datasets presented, this assumes a close-to-regular sampling, which may not always be the case. In this direction, a radius adaptive to the local sampling density would be more desirable in cases of variable sampling.

Furthermore, our method is not able to deal with the problem of double contours. Even if the scale of the noise is computed adaptively, each part of a badly registered scan will most likely have its own noise scale, and consequently, each scan will be reconstructed separately.

A further improvement could be done on the texture mapping step. For multiple-view datasets, we used a simple scheme in order to provide the mesh with texture extracted from the input images. However, due to illumination changes and registration errors, the texture on the triangles needs to be blended in order to achieve a continuous representation between neighboring triangles. There are many proposals in this direction [136, 85], but none of them have been applied underwater. Due to the aforementioned problems being very specific to underwater imaging, the seamless texture mapping problem also requires a tailored solution.

Chapter 6

Splat-based Surface Reconstruction

6.1 Introduction

While the method proposed in the previous chapter has proven to be useful in our area of application, we identified some drawbacks that should be solved.

First, the number of intersection queries during the RDT Delaunay refinement meshing process is quite high. This added to the fact that the intersection test requires the iterative RANSAC part, dooms the method to be quite slow. Therefore, alleviating the intersection computation cost is direly needed.

Second, in some cases the scale of the noise can be considered as constant, or at least a worst-case estimate may be enough to represent it. This is the case with our scenario, where the camera needs to be at a conservatively short distance from the object of interest in order to be able to obtain clear images amenable to reconstruction. When reconstructing the point cloud, this results in the retrieved samples having bounded noise values.

Additionally, even if the RDT surface meshing procedure allows the user to tune the quality measures on the retrieved surface, obtaining different parametrizations of the output surface would require running the whole algorithm again. In contrast, it would be desirable to maintain intermediate results between executions, to provide the user with the ability to test different configurations regarding the quality of the output surface mesh without having to perform all the computations again.

In this chapter, we introduce a method able to cope with the aforementioned problems. First, a splat-based representation is computed from the point set. A robust local 3D RANSAC-based procedure is used to filter the point set for outliers, then a local jet surface (a low-degree surface approximation) is fitted to the inliers. Second, we extract the reconstructed surface in the form of a surface triangle mesh through the RDT Delaunay refinement. In the present case, intersection queries are solved from the set of splats through a 1D RANSAC procedure.

6.2 Overview and Contributions

Our method is based on computing a global surface approximation using local surfaces. Instead of producing a consistent global representation of the surface through a memory-intensive global solver , such as solving for a signed distance or indicator function, and then using an isosurface meshing approach, our approach performs the merging of the different local surfaces at the meshing step.

We denote as *splats* these local surfaces, which may not be just planar but higher-degree approximations instead. A splat, computed using a local neighborhood of the input points, takes into account the fact that the input point set may be corrupted with noise and outliers. From the local neighborhood of an input point, we use the RANSAC method to determine which points are most likely to be a part of the surface. The splat is then computed using least-squares fitting of a differential jet. Thus, the differential jet approximation handles the effects of noise in the data, while RANSAC ensures using only inlier data in their computation.

The set of splats provides a global approximation of the shape sampled, amenable to coarse-to-fine meshing through Delaunay refinement [30]. The main advantage of the Delaunay refinement method is that it only requires computing intersections between line segment queries and the surface to be meshed. In our case, the surface-segment intersection queries are robustly solved by computing splats-segment intersections and running another 1D RANSAC procedure along the segment query. Even if small inconsistencies are present in the splat representation, RANSAC takes advantage of the redundancy between splats so as to be able to answer correctly the intersection query required by the surface meshing method. Figure 6.1 depicts an overview of our method.

It is worth noting that coherent orientation between splats is not required, since only intersection points are considered during the meshing regardless of their orientation. Thus, the method works on raw point sets, avoiding the requirement of per-point normals in most of the approximation-based methods of the state of the art. This makes the method amenable to point sets coming from a wider range of sources.

In addition to a low memory footprint, our two-step proposal allows us to generate the final mesh at different resolutions given the same splat-based representation by only changing the parameters of the meshing algorithm. Besides, the splat-based procedure allows the user to select the fitting degree of the splat as a means to trade smoothness for

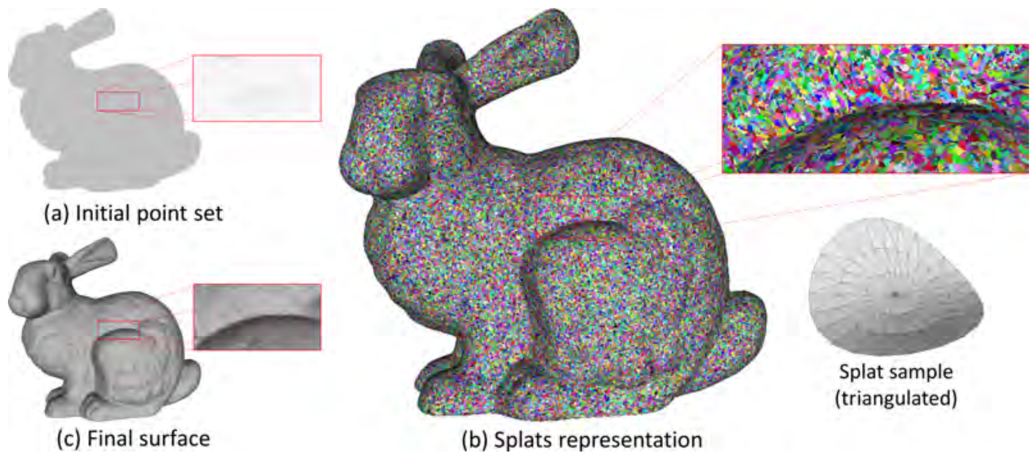


Figure 6.1: Overview of our method. (a) Input raw point set. (b) Splat representation (discretized using triangle fans for visual depiction). (c) Output surface triangle mesh generated through RDT Delaunay refinement.

fitting accuracy to the input data.

Finally, because of the limited support of the splats, this method does not find a surface over areas that are not sampled, allowing the recovery of surfaces with boundaries. Although hole filling capabilities are desirable in some cases, guessing the surface of large, undersampled areas often leads to artifacts on the final surface.

This chapter is structured as follows: Section 6.3 details the creation of the splat-based representation from the input point set. Then, Section 6.4 describes the surface meshing step through robust merging of the splats. Next, Section 6.5 illustrates the robustness and versatility of our method. Finally, Section 6.6 presents some conclusions and future work.

6.3 Creating the Splats

Given the input point set P, we compute a splat-based representation in which each splat is a local approximation of the surface. In its simplest form, a splat is a disk tangent to the surface and with a radius adapted to the local density of the point set. However, our method allows higher-degree approximations through so-called *jet surfaces*. We next explain how these jet surfaces are computed and how we combine them with RANSAC in order to achieve robustness to outliers.

6.3.1 Local Jet Surfaces

For each point $p_i \in P$, we pick its k nearest neighbors $K(p_i)$. In each local neighborhood we then compute a local smooth jet surface as introduced by Cazals and Pouget [45]. A jet surface is defined as a least squares approximation of a height function in a local reference frame. Coherence between neighboring splats is achieved through overlapping neighborhoods, and robustness to noise is achieved through least squares approximation.

More specifically, given a subsample of the points $K(p_i)$, and a local coordinate framework, the jet surface is defined as the Taylor expansion of a height function (truncated to a given degree). It is represented as:

$$z(x, y) = J_{B,d} + h.o.t., \tag{6.1}$$

where $h.o.t.$ stands for higher order terms, and $J_{B,d}$ is a jet surface of degree d (a d-jet) corresponding to:

$$J_{B,d} = \sum_{k=0}^{k=d} \left(\sum_{i=0}^{i=k} \frac{B_{k-i,i} x^{k-i} y^i}{i!\,(k-i)!} \right). \tag{6.2}$$

Such a local representation of a smooth surface is valid as long as the z-axis of the local coordinate framework is not included in the surface tangent plane. It may be computed from $K(p_i)$ by least square fitting. To get an accurate estimate of the jet surface at p_i we use a local coordinate framework obtained from $K(p_i)$ by principal component analysis (see Figure 6.2(a)).

From the jet surface we extract another representation, referred to as the *Monge form*. The Monge form is the Taylor expansion of the height function in the coordinate system whose axis are aligned aligned with the normal and the principal curvature directions of the surface.

$$\begin{aligned} z(x, y) &= \tfrac{1}{2}(k_1 x^2 + k_2 y^2) \\ &\quad + \tfrac{1}{6}(b_0 x^3 + 3b_1 x^2 y + 3b_2 x y^2 + b_3 y^3) \\ &\quad + \tfrac{1}{24}(c_0 x^4 + 4c_1 x^3 y + 6c_2 x^2 y^2 + 4c_3 x y^3 + c_4 y^4) + h.o.t. \end{aligned} \tag{6.3}$$

In the original method [46], this Monge form is used to evaluate differential properties of the surface at a query point, such as principal curvatures and directional derivatives. In our framework, we use it because it describes the local surface using fewer terms and will reduce the cost of computing intersections with segments during the Delaunay refinement meshing phase. Observe indeed that for a jet surface of degree $d = 2$, the number of coefficients in the jet surface is a priori $N_c = (d+1)(d+2)/2 = 6$, while the Monge form involves only two coefficients.

We now point out the difference between our approach and the MLS surfaces, since both approaches involve local computations around a query point. The second part of the projection operator presented by Alexa et al. [5] is also a local bivariate surface computation. However, the jet surface definition is explicit: it defines a fixed surface around a given query point, while MLS projection operators vary depending on the query point, which requires an iterative procedure for each intersection query with line segments. Section 6.4 explains how our framework takes advantage of the fact that the intersection query between a segment and a local surface of degree up to 2 can be computed more directly.

6.3.2 Outlier Rejection

For outlier rejection we use an approach based on 3D RANSAC [79]. The RANSAC method computes models from data corrupted with large quantities of outliers. At each iteration, it instantiates a model using a random sample of u points, where u is the minimum number of points needed to compute the model. All remaining points are tested against the current model, i.e., they are considered to be compatible with the model when they are within a maximum tolerance error from the model. Each point agreeing with the model votes so as to consolidate a consensus defined by the number of points agreeing with the model. After repeating the process a number of iterations, the model having largest consensus (support) is selected and a refined least-squares solution for the model is computed from all agreeing data. The intuition is that models generated from points that include outlier data should have little (weak) support while those generated from inlier data should have greater support.

In our algorithm, the jet surface computation is used as the model that RANSAC fits to $K(p_i)$. At each iteration inside the RANSAC procedure, a minimum set of points from $K(p_i)$ is selected and a model is computed from that. Then, the votes for this model are the number of points closer than δ_r from it. Note that since the surface can be of an arbitrary degree, we use the algebraic distance instead of the common Euclidean one, provided that it is more efficient to compute. Furthermore, a minimum number of inliers (δ_m) is required to accept the computed jet, otherwise the model is too weakly supported, and thus the points defining it should be considered as outliers. At the end of the RANSAC method, if a point p_i does not fit the best local jet (with the largest support), or the computed jet is too weakly supported, it is considered as an outlier and no further processing is carried out.

We will use a probabilistic approach to determine the number of iterations RANSAC needs to generate a model. As it is computationally too expensive to try each possible set of samples to build a model, we instead aim at finding the minimum number of iterations (N_{iter}) that ensures that, with probability q, RANSAC will pick at least one sample of size s free from outliers. If we define w as the probability that any point inside the currently selected sample is an inlier, the probability of this point being an outlier is $\epsilon = 1 - w$ [103]. Thus, the number of iterations is bounded by:

$$N_{iter} = \log(1-q)/\log(1-(1-\epsilon)^u). \tag{6.4}$$

We choose to fix $q = 0.99$ in our experiments. Since we do not know beforehand the percentage of outliers inside a given $K(p_i)$, we initialize ϵ with a worst-case estimate, which has been fixed to $\epsilon = 0.5$ for all experiments shown. Then, at each iteration, we update this estimate if the current computed model has greater support. The minimum number of points required to generate the jet surface depends on the degree of the jet. The number of coefficients c of a jet surface, and consequently the minimum number of points for the fit, is defined by $N_c = (d+1)(d+2)/2$.

From the $K(p_i)$ set of points, the RANSAC procedure extracts its subset of inliers, which are referred to as $M(p_i)$. In a final step, RANSAC computes the final jet-surface using all the points in $M(p_i)$.

6.3.3 Splat Sizing

Although each of the splats has finite support, we favor redundancy between neighboring splats since we take advantage of this property during meshing. The size of the splat is simply computed from $M(p_i)$ as the mean distance from each sample $p_j \in M(p_i)$ to p_i so that the size of the splat depends on the local point set density.

6.4 Meshing

We have generated one splat for each input point where RANSAC was successful, and because of the redundancy between them, the same area may be covered by more than one splat. In order to obtain the final surface triangle mesh from the splats, we use again the coarse-to-fine meshing algorithm based on the concept of RDT and Delaunay refinement [30]. Thus, as we did in the previous chapter, we basically need to infer a line intersection test for our splat representation.

Calculating the intersection between a line segment query and a jet surface of $d > 1$ has a high computational cost. For this reason, we use a planar disk version of the splat as the first approximation for efficient intersection detection. This planar approximation is defined by the origin of the splat and the normal at this origin (both defined by the computed Monge form of the jet surface), and the radius computed in section 6.3.3.

Prior to meshing, we insert all the disks into a hierarchical AABB tree data structure (Axis-Aligned Bounding Boxes) in order to further accelerate the intersection detection. Once we detect the disks intersected by a segment query, we proceed as follows for each disk. We move the intersection points along the segment query in order to refine the intersection points with the associated local higher degree jet surfaces. Note that when

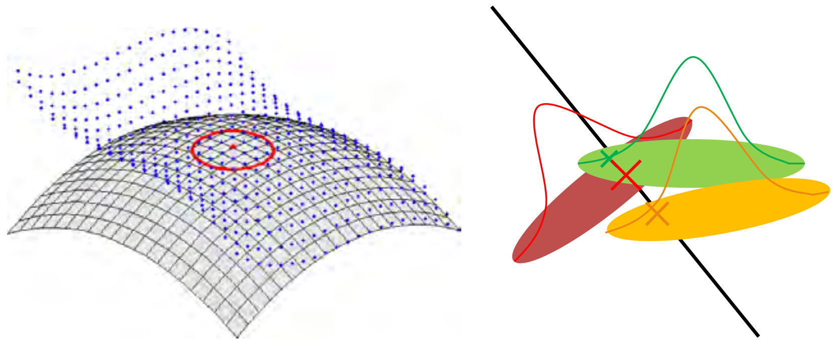

(a) (b)

Figure 6.2: Intersecting a splat with a line segment. (a) A jet surface (in gray) is fitted to the neighborhood of the red point. The limit of the splat support is depicted with a red circle. Blue points depict the input points. Observe how the approximation of the splat is faithful closer to its center. (b) Several disks are intersected by the query segment. The contribution of each intersection point is weighted by a function of its distance from the center of the splat. The lines over the disks depict the Gaussian weights used.

the local surface has degree $d = 2$, there is a closed form solution [143], and local surfaces of degree $d > 2$ require solving a non-linear minimization along the segment. Starting from the intersection point with the disk approximation, the intersection point is moved towards the first intersection point along the segment using Levenberg-Marquardt optimization. We observe that higher order local surfaces are useful in undersampled datasets in which local changes in the surface are more abrupt and thus require more complex local surfaces. However, we also observe that using local surfaces of high degree produces surfaces that capture the noise instead of correcting it.

For initialization an initial subset of the centers of the computed splats, 20 in all results shown, are used to create the initial triangulation required to seed the mesh refinement process.

6.4.1 Merging Local Intersections

If the input point set is dense enough, it is very likely that each intersection query returns not just one but a set of intersection points. Assuming for the moment that there are no outliers among these intersection points, we need to provide a single intersection point summarizing the local splats intersected. We first observe that the local surface computation is "variability centered", i.e., more reliable when close to the center of the splat. We compute a weighted mean to reflect this observation (Figure 6.2(b)).

Each splat is assigned a Gaussian weight function located at its center. Each intersection point is then given a weight according to its distance from the center of the splat:

$$w(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}, \tag{6.5}$$

where $\sigma = r_i \gamma_g$, and r_i denotes the radius of the splat i (its size) and γ_g denotes a user parameter used to adjust the scale of the Gaussian. The final merged intersection point p_{int} is computed as:

$$p_{int} = \frac{\sum_{i \in Q} w(\|p_i - c_i\|) p_i}{\sum_{i \in Q} w(\|p_i - c_i\|)}, \tag{6.6}$$

where Q denotes the set of intersected splats, and p_i and c_i denote the intersection point and the origin of the intersected splat, respectively.

6.4.2 Robust Intersection Query

Despite our efforts for achieving coherence between neighboring local splats, the weighted averaging process described above is not appropriate for high levels of noise. In addition, the local approximations of the surface are computed from a given center (an input point), thus their accuracy decreases with their distance from this center. Furthermore, depending on parameter δ_r, the RANSAC procedure described above is not able to detect outliers that are far from the true inlier points yet close to the jet surface approximation. For these reasons, we need an intersection test robust to both low-conformity among neighboring splats and remaining outliers.

We have one splat for each input point, hence, due to splat overlaps, the surface is locally represented by more than a single splat. Intuition tells us that, in general, most of the splats are local faithful approximations of the surface, except for a few outliers. Figure 6.3(a) depicts this general configuration: most intersection points are within one cluster, very close to one another, while outlier intersection points are isolated. We thus aim at extracting the final intersection point by taking into account the cluster of points with the largest support.

In order to extract this cluster, a 1D RANSAC procedure is applied along the intersection query where the model is a weighted centroid. At each iteration of the RANSAC procedure, two points are randomly selected, the minimum to define a centroid, and a model is instantiated from them. Then, we rank this model according to the number of points being closer to the centroid than a distance parameter δ_i. Two sample iterations of the algorithm are illustrated in Figures 6.3 (b,c). At the end, the centroid having largest support is selected, providing the cluster of points used to compute the weighted averaging as described above. A percentage of the length of the segment query is used to define the δ_i parameter.

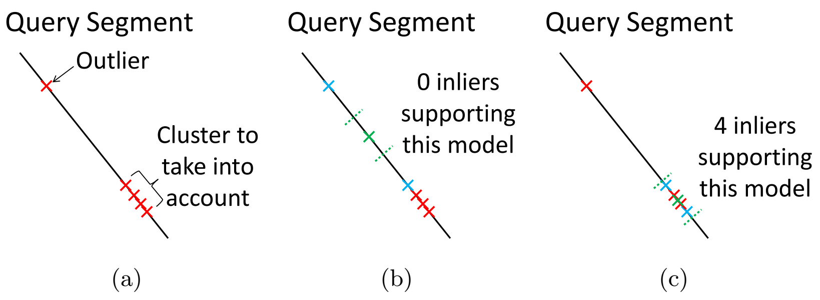

Query Segment
Outlier
Cluster to take into account
(a)
Query Segment
0 inliers supporting this model
(b)
Query Segment
4 inliers supporting this model
(c)

Figure 6.3: Robust intersection. (a) The splats that are coherent among each other generate a cluster of intersection points, while a non-coherent splat generates an isolated intersection point, classified as an outlier. (b) and (c) illustrate two possible iterations of the RANSAC algorithm, centroids (green) generated from outliers have weaker support than the ones generated from inliers.

Note that this procedure requires at least two splats overlapping a given area. If a single splat is intersected, we return no intersection as no consensus can be established locally. Redundancy among splats is thus central to provide robustness in our approach.

6.5 Results

We implemented our method using components from the CGAL library [1]. Our current implementation is sequential. Table 6.1 lists all the parameters used by the two steps of our algorithm. Note that some parameters are not critical and thus have been set once for all experiments shown. The data-dependent parameters are k (nearest neighbors) and those driving the splat-RANSAC procedure. There are 3 additional parameters required for meshing. All the parameters used to generate the results are listed in Table 6.3, along with the running times for the two steps of the algorithm. Note that we choose similar values for α_r and α_d, because this leads to surfaces with triangles of similar size.

As illustrated in Figure 6.4, our method works well in the ideal scenario, with no noise or outliers. The Stanford Bunny dataset is chosen to illustrate the behavior of our method under harder noise conditions (Figure 6.5). Figure 6.6 (a) confirms that our method is already able to deal with noise with local surfaces of degree 1. The other sub-figures show the results obtained when increasing the degree. The timings listed in Table 6.3 confirm a rapid decrease in performance for degrees higher than 2, without visible increase in the quality of the results. In the remainder we report results only for local surfaces of degree $d = 2$.

Robustness to unstructured outliers is illustrated in Figure 6.7. We artificially add outliers to the Stanford Bunny point set by adding uniform random points inside a loose bounding box (enlarged by 5% of its diagonal) of the original point set. The second row

Table 6.1: Parameters used by the two steps of our algorithm. We also specify the constant values used in all the experiments shown in Section 6.5. BBD denotes the Bounding Box Diagonal and QSL denotes the Query Segment Length.

Parameters		
Splats		
Name	Description	Default Value
k	Number of neighbors to take into account	(variable)
d	Degree of the splat fit	2
δ_r	RANSAC distance to plane threshold	$0.01BBD$
δ_m	RANSAC minimum number of inliers	(variable)
Meshing		
Name	Description	Default Value
α_a	Surface meshing angle bound	10
α_r	Surface meshing radius bound	(variable)
α_d	Surface meshing distance bound	(variable)
δ_i	RANSAC threshold for the intersection query	$0.05QSL$
γ_g	Gaussian deviation factor	0.25

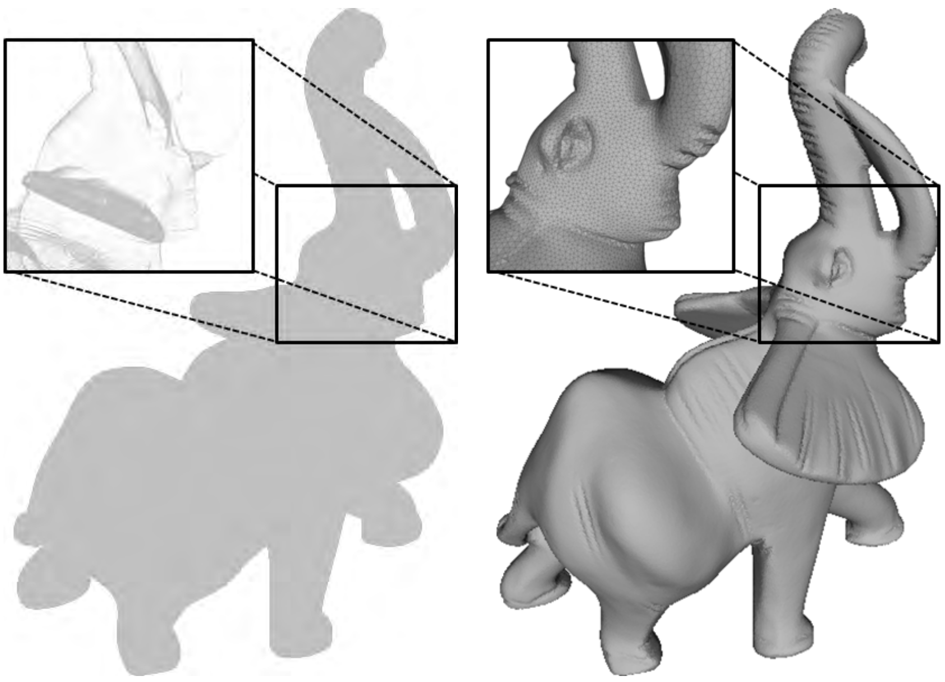


Figure 6.4: Elephant dataset: 1,537,283 points, both noise and outlier free. Left: input pointset with closeup. Right: reconstructed surface with closeup.

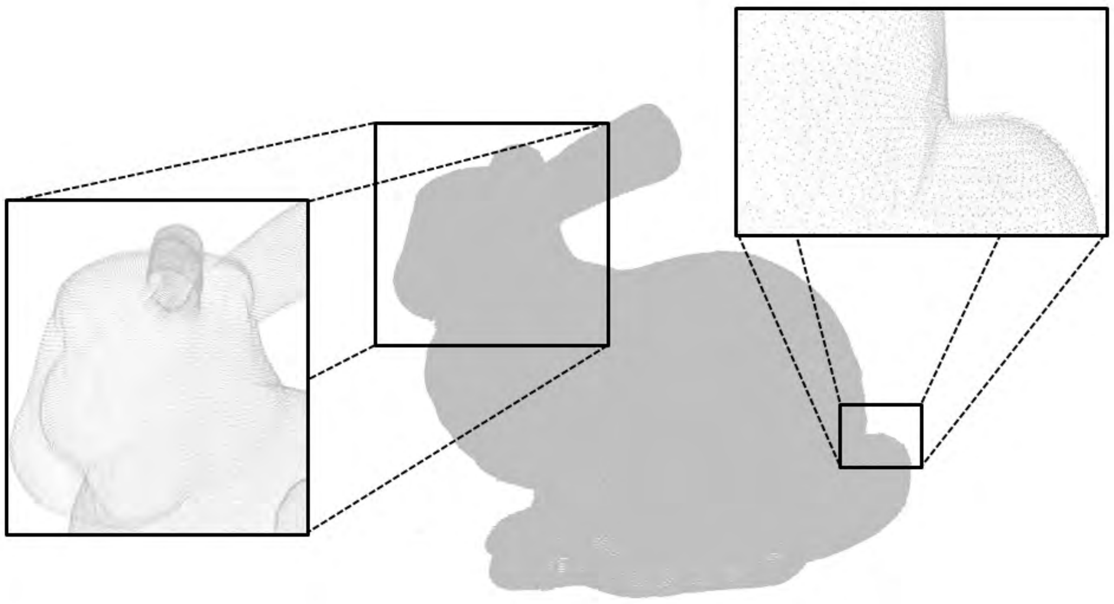
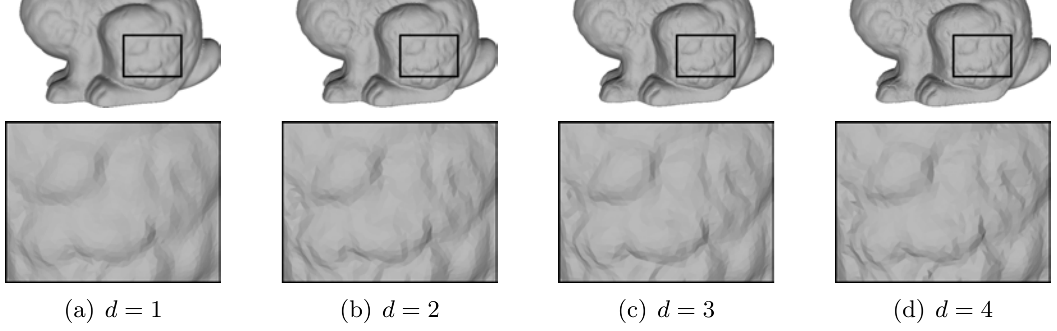


Figure 6.5: Stanford Bunny: 362,272 points with a high level of noise.



(a) $d = 1$ (b) $d = 2$ (c) $d = 3$ (d) $d = 4$

Figure 6.6: Stanford Bunny. Reconstructed surfaces with a gradual increase of the degree of the splats.

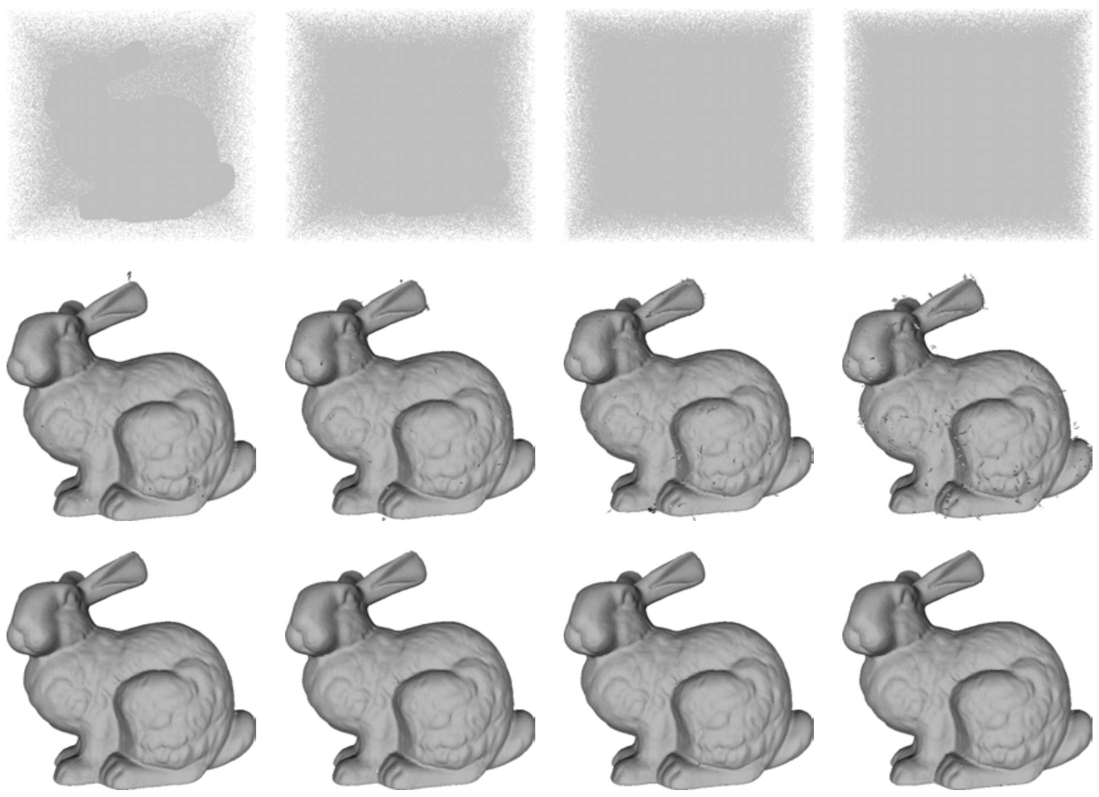


Figure 6.7: Robustness to outliers. The first row shows the outlier-ridden point sets. The percentages of outliers added, with respect to the number of points on the original point set, are 50%, 100%, 150% and 200%, added inside its bounding box (enlarged by 5%). The last 2 rows show the results when applying our method to its vertically corresponding point set. The second row uses $\delta_r = 0.005BBD$, while the third row uses a more restrictive $\delta_r = 0.0025BBD$.

shows the results when applying our method with a less restrictive δ_r. In the presence of outliers, this parameter becomes relevant as it is the one defining the inlier set at each RANSAC iteration. Using a looser threshold leads to outlier points close to the surface, which generate outlier splats. These outlier splats produce holes in the final surfaces since they are not coherent with the other splats, and they are not detected for intersection by our robust intersection query. The third row in Figure 6.7, where δ_r is set to be more restrictive, shows that the results are quite similar for all outlier-ridden datasets. When taking a look at the running times of these datasets in Table 6.3, we note that there is a noticeable decrease in performance during the creation of the splats. This is due to the RANSAC procedure being unable to obtain a result with enough inliers to estimate the number of iterations ruled by Equation (6.4) to decrease. This means that RANSAC reaches the maximum number of iterations to decide finally that there is no valid jet surface possible at a given outlier point.

In order to measure the accuracy of our method for variable amounts of noise and

Table 6.2: Robustness on the synthetic sphere point set. The 1st column shows the added noise (standard deviation), the 2nd, the percentage of outliers added according to the original point set. The 3rd and 4th columns indicate the vertices and edges of the resulting surface. The columns from the 5th to the 7th show the mean/min/max error of the points on the surface from the original unit sphere, while the last two columns correspond to the number of non-manifold edges and vertices in the resulting surface.

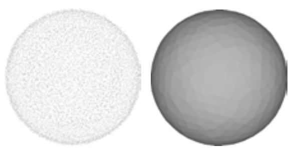
Noise	Outliers	Vertices	Edges	Mean Error	Min Error	Max Error	NM Edges	NM Vertices
0	0	840	1676	2.33e-05	1.86e-05	4.16e-05	0	0
0.01	0	829	1654	0.001438	1.86e-06	0.005201	0	0
0.01	25	818	1632	0.001620	3.80e-07	0.006418	0	0
0.01	50	837	1670	0.001926	8.77e-06	0.007822	0	0
0.01	100	824	1646	0.002120	3.25e-06	0.010432	4	0
0.025	0	830	1664	0.004195	6.16e-06	0.016708	16	0
0.025	25	830	1662	0.004322	1.10e-05	0.022721	12	0
0.025	50	821	1642	0.004567	2.17e-06	0.023205	8	0
0.025	100	829	1666	0.004980	1.27e-05	0.023553	23	0
0.05	0	864	1749	0.013898	5.82e-05	0.063856	117	8
0.05	25	868	1752	0.013898	1.51e-06	0.093498	123	11
0.05	50	868	1769	0.013716	1.93e-05	0.074861	162	11
0.05	100	864	1708	0.015326	1.04e-05	0.090198	133	16

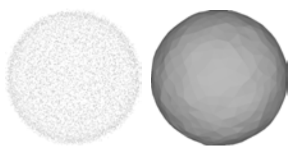
outliers, a synthetic dataset consisting of a unit sphere has been generated. Table 6.2 and Figure 6.8 show the results for added Gaussian noise and outliers. Table 6.2 shows that noise makes the accuracy of the final surface decrease incrementally and adds non-manifoldness caused by the non-conformity between neighboring splats. For a high level of noise, the method is not able to recover a good splat approximation and starts degrading. However, even in this case the effect of outliers is not noticeable.

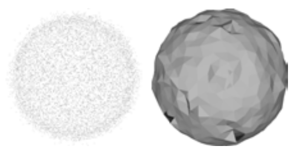
Despite a high resilience to noise and outliers, and given the local nature of the primitives used to approximate the inferred surface locally, our method is not always able to fill in holes in the resulting surface. This is bad in some cases where it would be desirable to fill small holes in the surface due to missing data, but it is also useful to get no reconstruction in some parts when the goal is to reconstruct surfaces with boundaries. The foot dataset depicted in Figure 6.9 illustrates this dilemma.

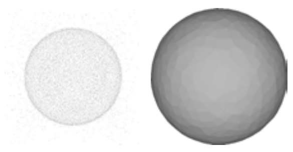
Our mesh generation procedure provides an easy way to obtain different resolutions for the final surface mesh by changing the RDT Delaunay refinement parameters (see Figure 6.10). Methods relying on variants of the marching cubes commonly require re-creating the voxel grid at the desired resolution in order to modify the properties of the output mesh, while our splat-based representation is unchanged.

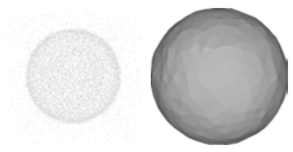
So far, the method has been tested against regularly and densely sampled datasets. Figure 6.11 shows the results of the method for a sparse dataset. After applying the method with $k = 25$, the resulting surface contains some holes; they can be seen in Figure 6.11 (b). This is due to undersampled areas. Observe that it makes no sense to apply the RANSAC method presented in Section 6.4.2 when there is only one intersected splat,

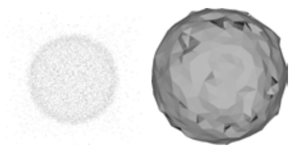

(a) N = 0.01 / O = 0

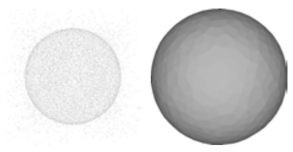

(b) N = 0.025 / O = 0

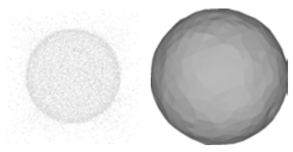

(c) N = 0.05 / O = 0

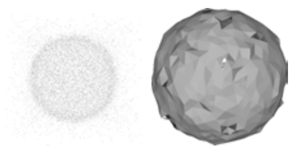

(d) N = 0.01 / O = 25

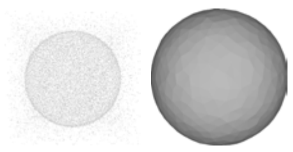

(e) N = 0.025 / O = 25

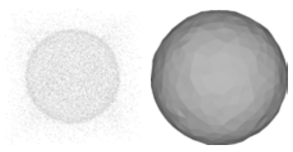

(f) N = 0.05 / O = 25


(g) N = 0.01 / O = 50


(h) N = 0.025 / O = 50


(i) N = 0.05 / O = 50


(j) N = 0.01 / O = 100


(k) N = 0.025 / O = 100

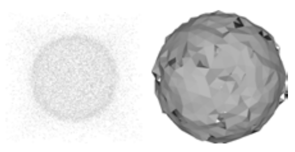

(l) N = 0.05 / O = 100

Figure 6.8: Synthetic unit sphere dataset. Results for increasing levels of noise and outliers. The original point set was made out of $10,242$ points. N stands for noise (standard deviation) and O stands for outliers (percentage). These results correspond to the data in Table 6.2.

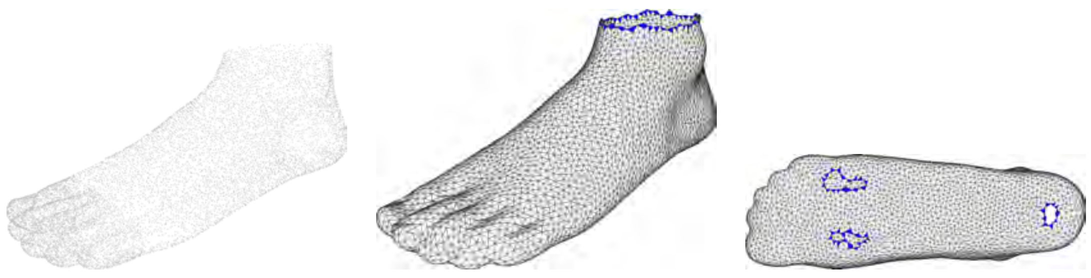


Figure 6.9: Foot dataset ($10,010$ points). The first column shows the point set. The second and third columns show two views of the model with boundary triangles depicted in blue.

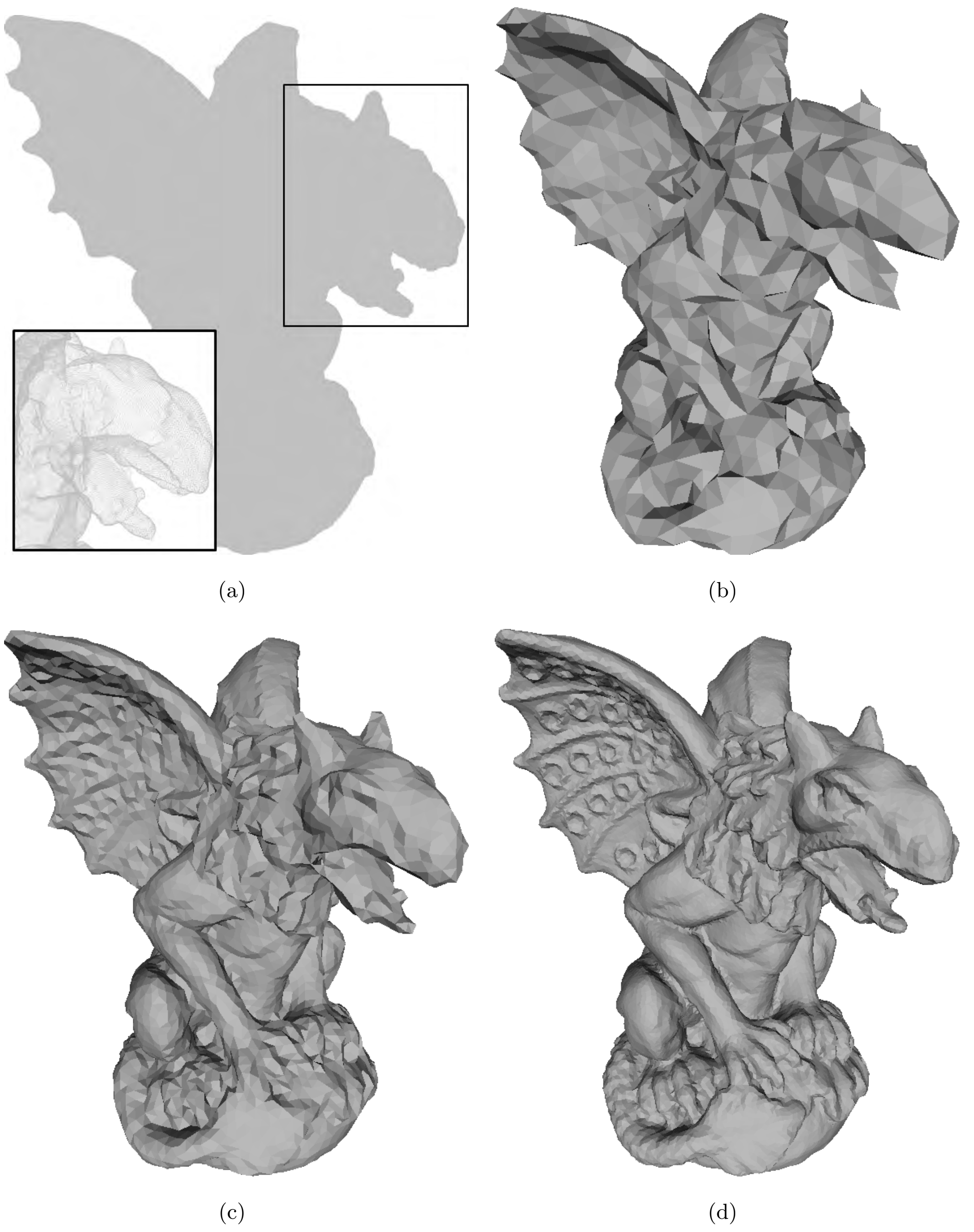


Figure 6.10: Different resolutions and quality of the final surface can be achieved by changing the parameters of the surface meshing without changing the splats approximation. (a) presents the point set (863, 210 points), while (b), (c) and (d) show incremental levels of resolution with respect to the size of the triangles and the approximation bounds.

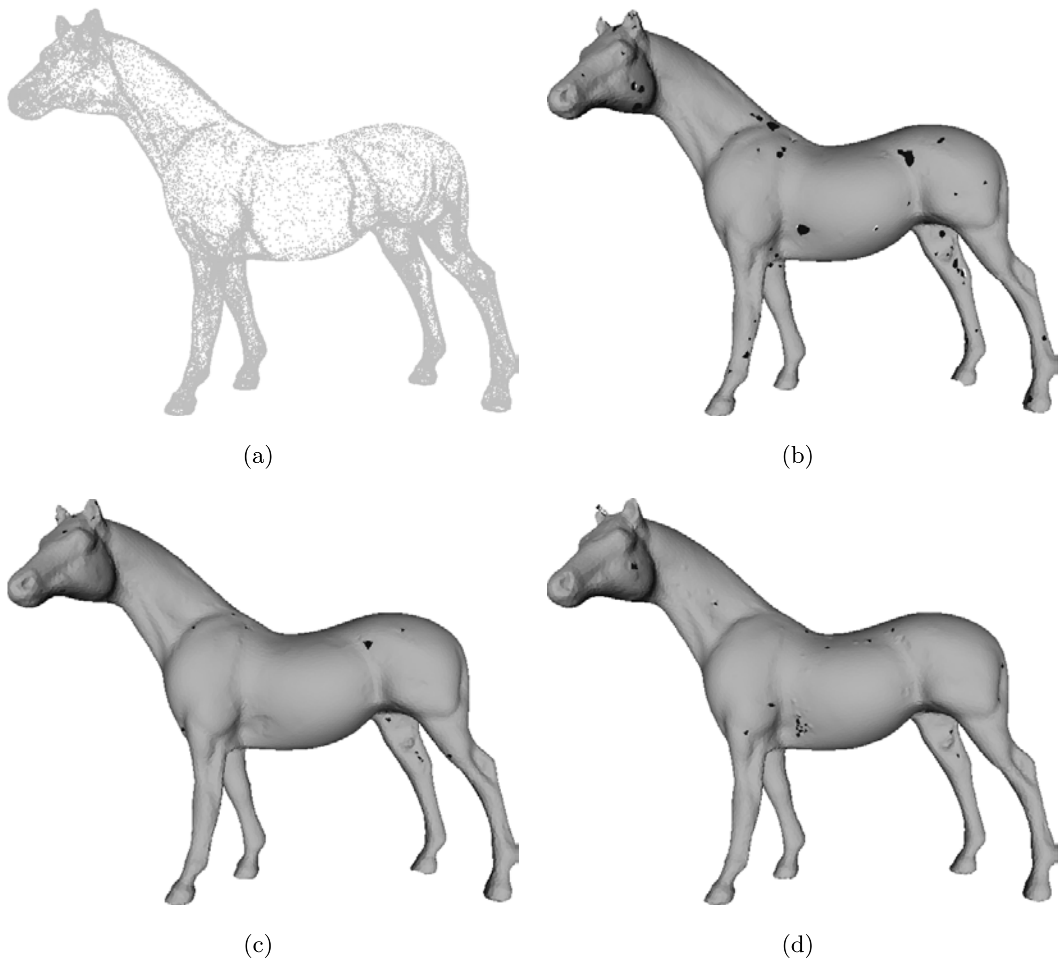
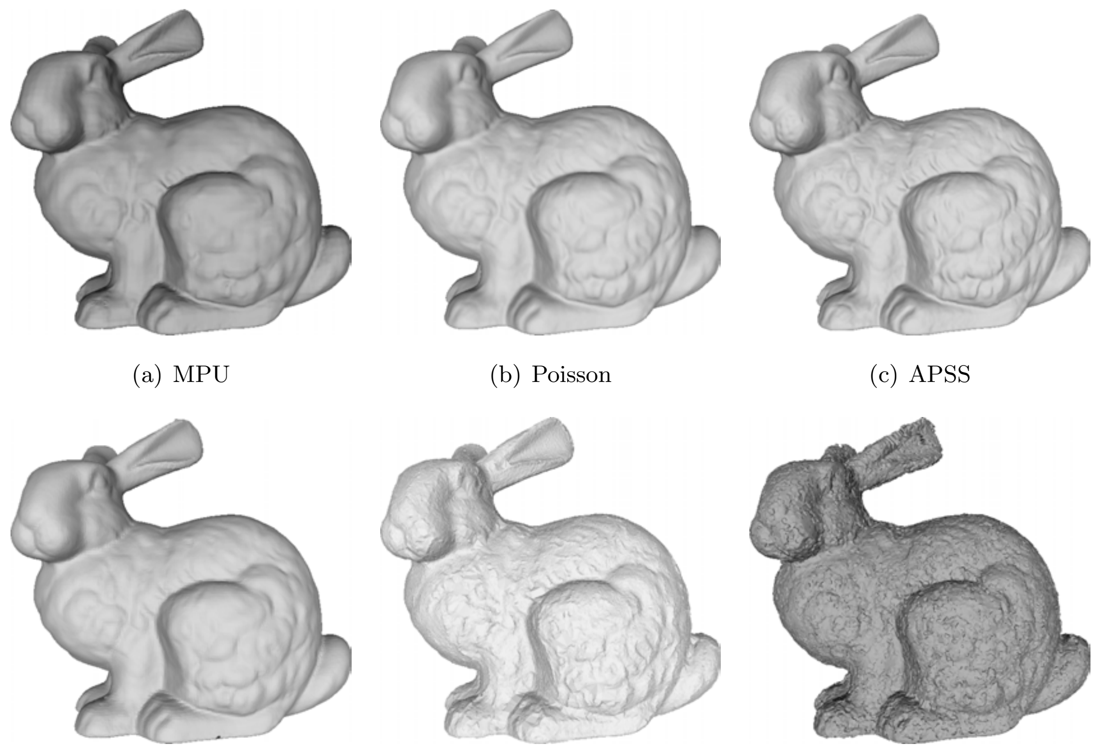


Figure 6.11: Horse dataset, (a) example of a very sparsely sampled point set, 100,000 points. We then show the results of our method for different values of the k parameter. (b) corresponds to $k = 25$, (c) to $k = 50$ and (d) to $k = 100$.

since it is impossible to tell whether this intersection point is an outlier or not. Thus, at least two splats are required to tell if an area is correctly represented (intersection points are close to one another), and three or more are needed to run the RANSAC procedure. One may think that increasing k would also make the resulting splats bigger and cover the holes. This is what happens in Figure 6.11 (c), when we use $k = 50$, although there are still some holes left. Note however that increasing k too much, as depicted by Figure 6.11 (d), leads to bad local fitting and hence bad fitting between neighboring splats. As a consequence, more holes are generated because of our RANSAC intersection test. This issue, left for future work, suggests that a fixed k may not be a correct solution for non-uniformly sampled datasets, and should be adaptive to sampling density.

Figure 6.12 shows how some of the state-of-the-art methods behave when applied to the Stanford Bunny dataset. They can be compared with the ones presented in Figure 6.6. We


(a) MPU (b) Poisson (c) APSS

(d) VRIP (e) Robust Cocone (f) Power Crust

Figure 6.12: Reconstructions for the Stanford Bunny dataset using some of the methods from the state of the art. (a) Multi-level Partition of Unity [165], (b) Poisson [118], (c) Algebraic Point Set Surfaces [97], (d) Volumetric Range Image Processor [60], (e) Robust Cocone [65] and (f) Power Crust [13].

observe that the interpolatory methods, Robust Cocone [65] and Power Crust [13], are not able to deal with noise. Regarding the methods based on approximating the surface, they all achieve smoother results. The results obtained by APSS [97], Poisson [118], VRIP [60] and MPU [165] are comparable, but MPU returns an over smoothed surface in some parts. Regarding the APSS method, it depends on the scale parameter. If this scale parameter is too low, it generates a fitting too tight to the input points and causes some noise to be captured in the surface. Our method recovers a surface very similar to that obtained by Poisson and VRIP, but does not require any additional information. Poisson, MPU and APSS require oriented normals at input points, while VRIP requires the knowledge of the individual scans forming the point set.

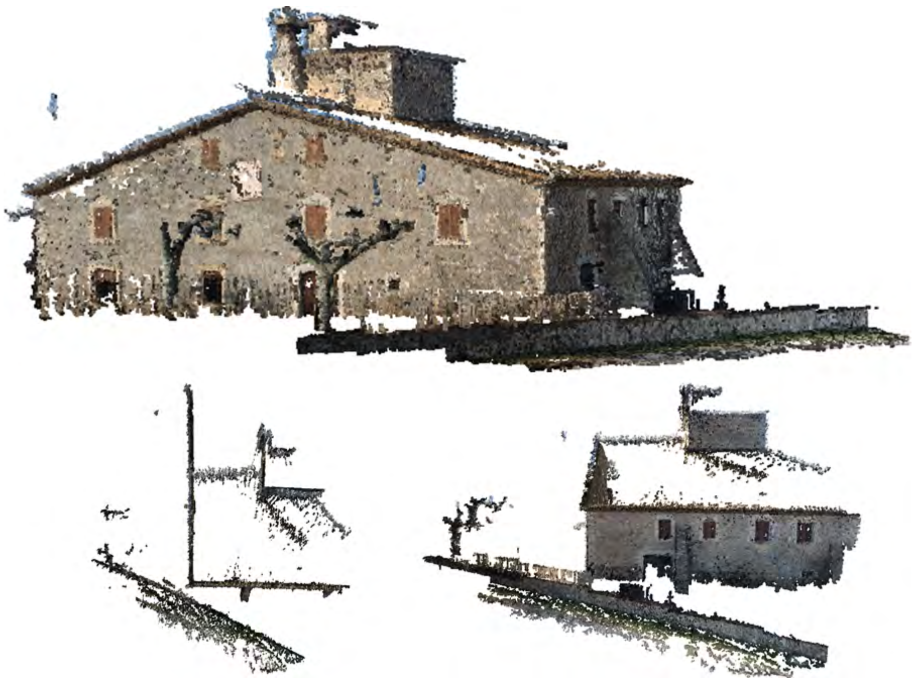
Finally, to test our method to its limit in terms of robustness, we ran it on point sets obtained from images (outdoor, aerial and underwater) through dense computer vision photogrammetry methods.

Depending on the method used to obtain the dense point set, the data has different properties. For this reason, we evaluate our method against the two approaches described in Section 2.6 for dense point set reconstruction. On the one hand, we use the Patch-based Multi-View Stereo (PMVS) method [84]. This greedy method has the advantage of providing a smooth point set, with low noise and outliers. However, its disadvantages include a lower coverage of the area (due to failures during propagation) and the creation of structured outliers (i.e., outliers are not single points, but clusters of points). On the other hand, we use a plane sweeping method [55, 209], where each image in the sequence contributes a dense depth map to the model. This fact induces a huge sampling rate for a given dataset. However, the lack of a coherence check between depth maps leads to the creation of a huge amount of both outliers and noise. Furthermore, if the cameras are not well registered, the model also contains doubled contours, that is, parts of the model that are repeated more than once, but their positioning does not match.

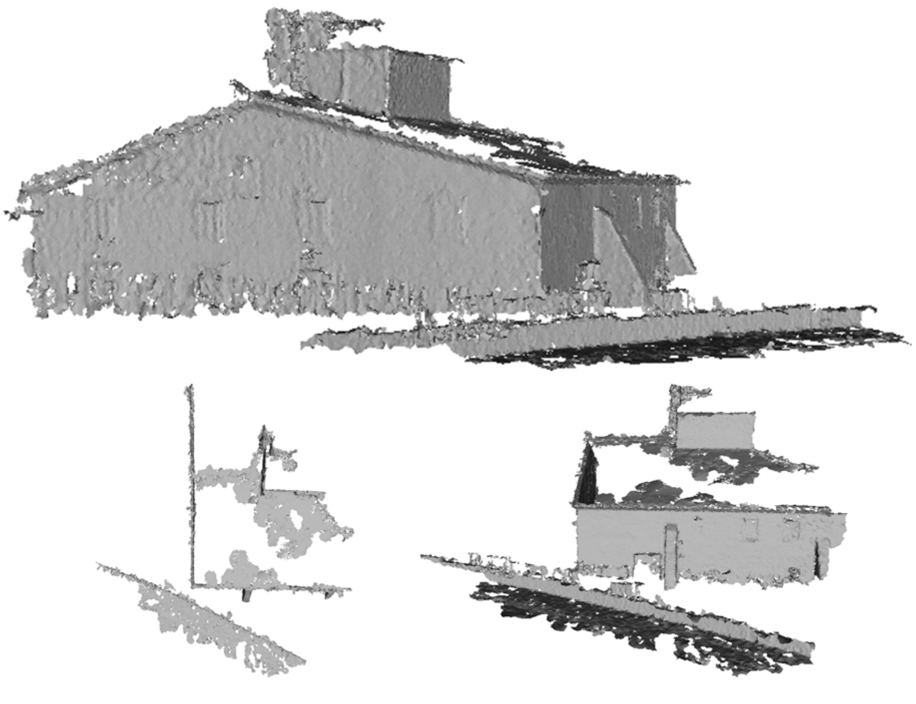
We start with two outdoor datasets generated using PMVS. The first one consists of an old Farm (Figure 6.13), and the second is the Fountain dataset from the multi-view stereo benchmark of Strecha et al. [201] (in Figure 6.14). As expected, the point sets are a very smooth representation of the object with few outliers. Consequently, our method has no problem in reconstructing both cases faithfully. Note in the case of the Farm, small structures like trees have disappeared because of their small support that made the splat generation step consider them as outliers.

In order to test the application of PMVS along with our surface reconstruction method on an underwater scenario, we use three of the datasets presented in the last chapter: *Shallow Water* (Figure 6.15), *Coral Reef* (Figure 6.16) and *Tour Eiffel* (Figure 6.17). We have seen that both the Shallow Water and Coral Reef datasets are examples of areas observed from a downward-looking camera. The results of the Shallow Water dataset in Figure 6.15 show a noise-free point set with few outliers (black points). Our method is able to recover the surface of these areas without any further difficulties. For the Coral Reef dataset in Figure 6.16, the resulting point set is also smooth, but the fact that the camera was held at a constant altitude lead to the small details in the scene to be undersampled. For this reason, the small mound-like parts in the middle of the shape have not been recovered correctly, and several non-manifold parts were produced. Nonetheless, the rest of the surface is correctly recovered.

A key aspect of the PMVS method in front of plane sweeping is that it provides per-point normals. Thus, the results of this method are suitable for application on surface reconstruction algorithms requiring an oriented point set as input. Consequently, we tested the Tour Eiffel dataset not only with our method but also with the Poisson [118] and APSS [97] methods, both requiring this type of input. Figure 6.17 shows these results, where we can observe that Poisson provides a smoother closed surface, omitting some of

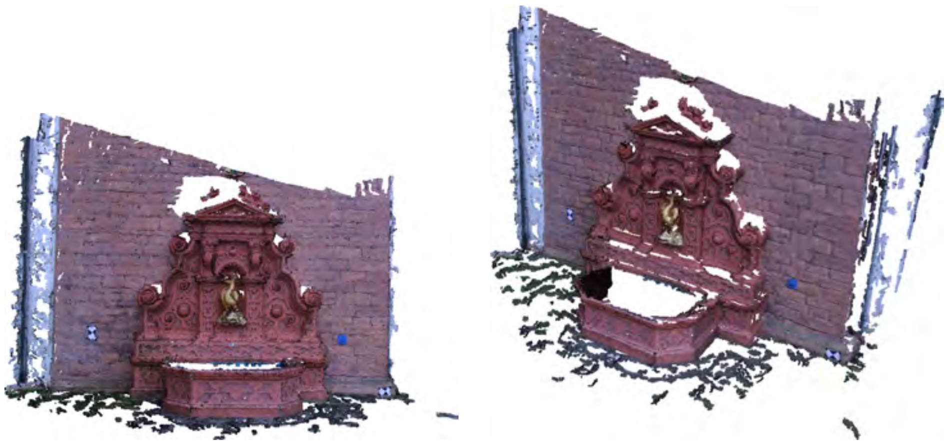


(a) Point Set



(b) Surface

Figure 6.13: Farm dataset, 1,136,957 colored points, obtained using PMVS [84]. (a) shows the point set, (b) presents the surface. While the overall structure is recovered, small structures like trees are eliminated.


(a) Point Set

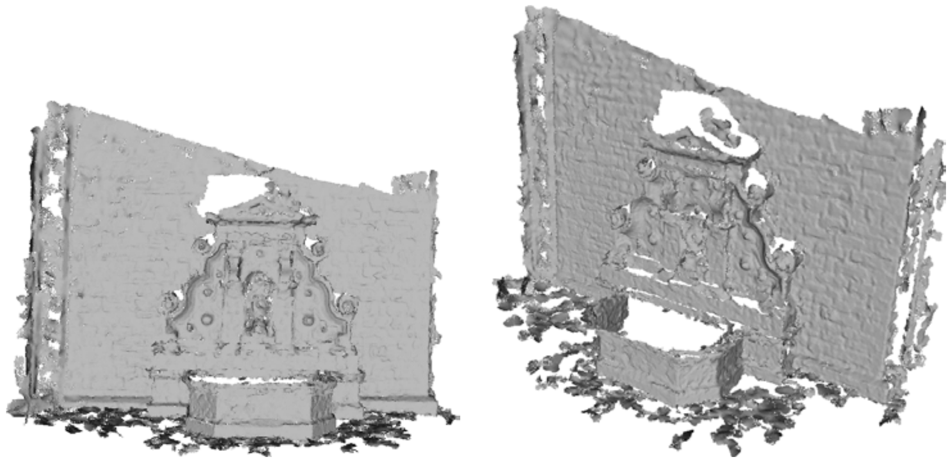
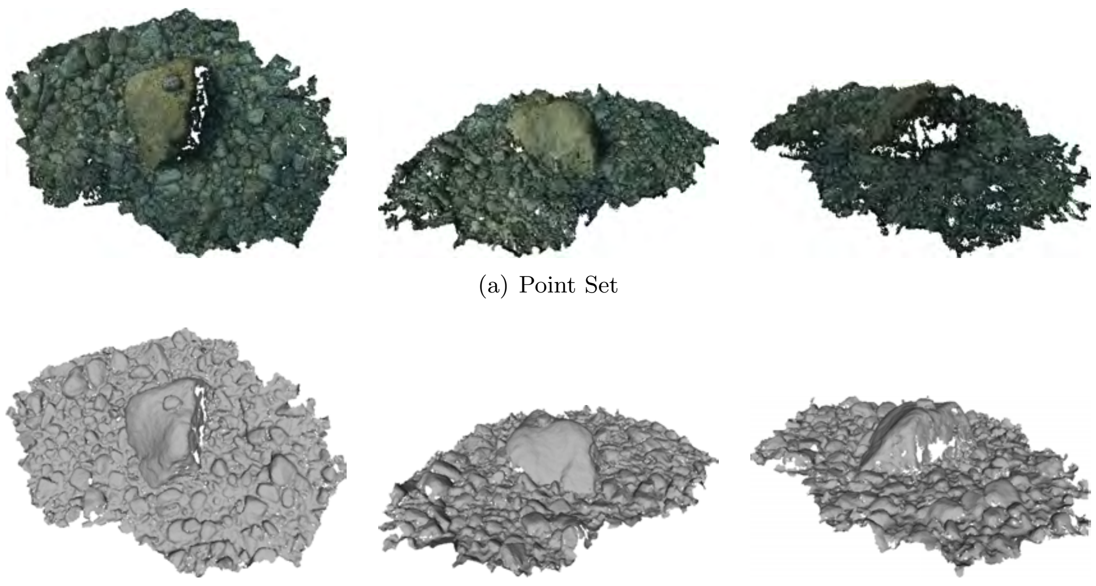

(b) Surface

Figure 6.14: Fountain dataset, 1,886,489 colored points, obtained using PMVS [84]. (a) shows the point set, and (b) the reconstructed surface.


(a) Point Set

(b) Surface

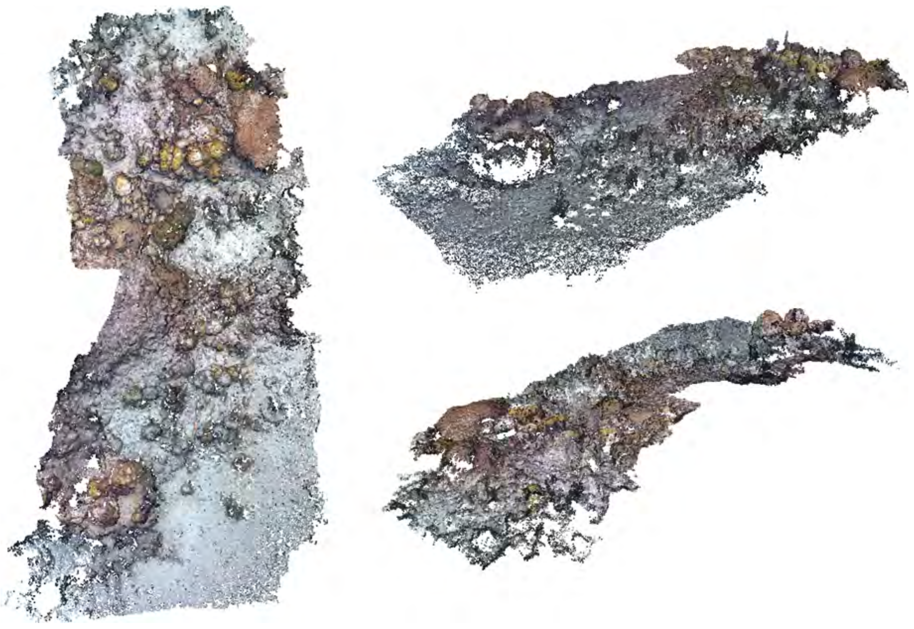
Figure 6.15: Shallow Water dataset, 863,162 points, (a) obtained through PMVS [84], and (b) the reconstructed surface using our method.

the details of the shape. On the other hand, the APSS is able to recover a bounded surface when there is a lack of sampling on the dataset, and fits tighter to the input points, thus preserving the details of the surface. Finally, our method yields a mixture of both results, by recovering a bounded surface and obtaining a smooth surface a bit more detailed than that of Poisson.

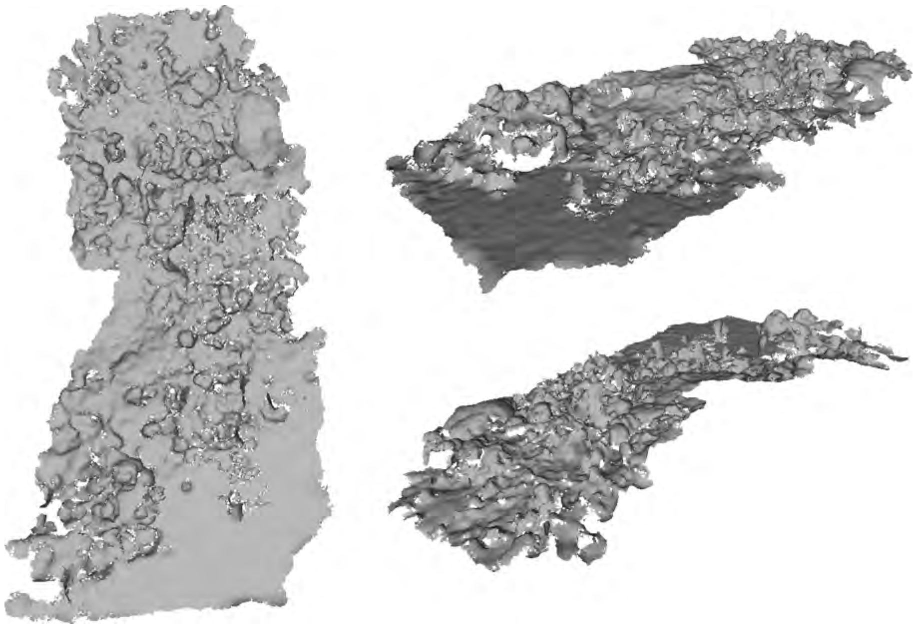
After overviewing the results with the smooth point sets provided by PMVS, we test our method against point sets retrieved using a plane sweeping algorithm. Due to the large quantity of noise and the high number of outliers they present, these point sets pose a greater challenge to our method. Furthermore, many of the outliers are structured (e.g., they are located on the same plane), and not uniformly spread as assumed previously.

Starting again with a pedestrian example, Figure 6.18 shows the results obtained with a column capital. Despite the structured outliers present in the input data, our method is able to remove the vast majority of them, generating a few wrong splats in the areas near the inferred surface, which then translate into wrong off-surfaces.

Figure 6.19 depicts a large scale aerial measurement of the abbey of Cluny (France). The point set presents even more difficulties, since in addition to the large number of structured outliers, the sampling is both widely variable and anisotropic. These differences in sampling are due to the variable distance from the buildings to the camera: buildings closer to the camera are reconstructed with higher resolution and vice-versa. Given this non-uniform sampling, we use a small neighborhood parameter k to recover parts of the

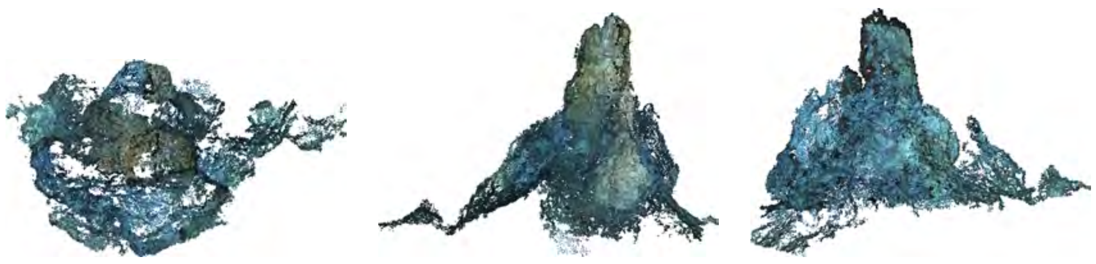


(a) Point Set

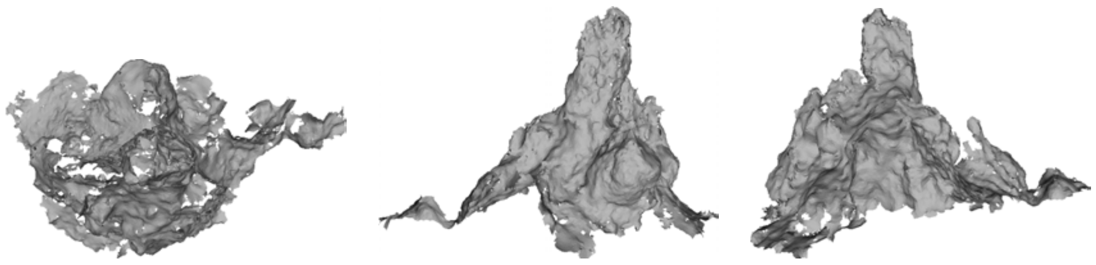


(b) Surface

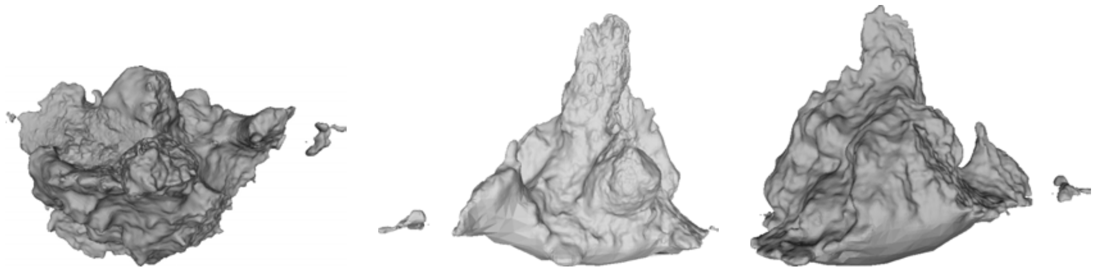
Figure 6.16: Coral Reef dataset, 215,809 points, (a) obtained through PMVS [84], and (b) the reconstructed surface using our method.



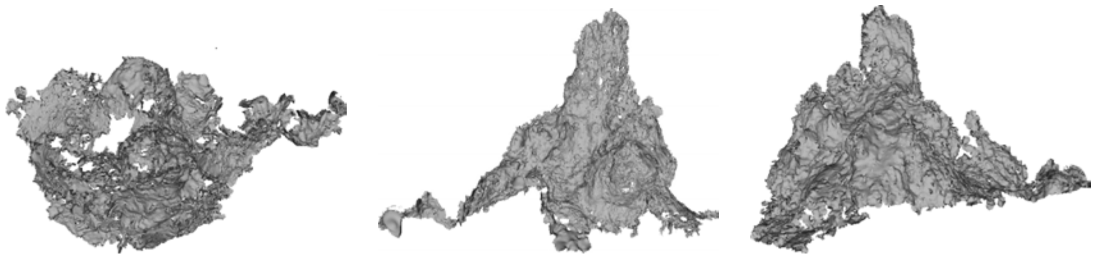
(a) Point Set



(b) Our method



(c) Poisson



(d) APSS

Figure 6.17: Tour Eiffel dataset, 168,502 colored points, (a) obtained using PMVS [84]. We used the point set in (a) with our method (b) and with Poisson [118] (c) and APSS [97]. One can observe that our method provides a smooth representation like Poisson, while recovering boundaries as with APSS.

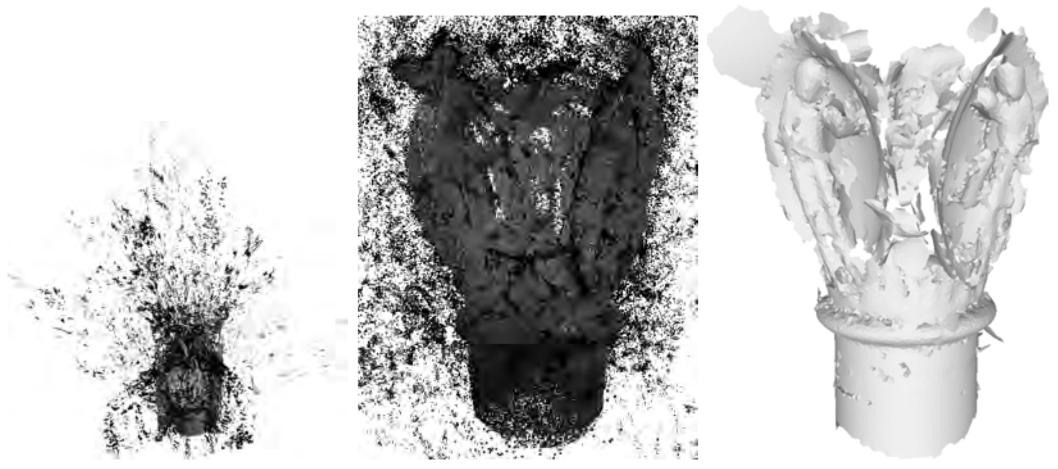


Figure 6.18: Column Capital optical dataset, 419,488 colored points. (a) Shows the full set of input points, (b) a closeup of the inlier area and (c) the surface obtained by our method.

surface with few representative points. However, as using a small neighborhood is less robust to outliers, we need to enforce the splats for greater support by increasing parameter δ_m. Figure 6.19 (c,d) illustrate the reconstructed surfaces when using a small support ($k = 50$, $\delta_m = 25$): our method is able to reconstruct parts at the back of the scene but also creates many wrong small surfaces at the front (the church building), where the point set contains many structured outliers. Figure 6.19 (e,f) illustrate how our method achieves better robustness when enlarging the support of the splats (same k, but $\delta_m = 40$) at the price of fewer parts of the scene covered in the reconstruction. This suggests that the two parameters k and δ_m are our means to trade robustness for coverage. An automatic parameter selection for adjusting the neighborhood k to the local sampling density is left for future work. Finally, note that since our method seeks a smooth surface, the sharp creases of buildings are not accurately reconstructed.

Regarding underwater datasets, we test the previously presented for PMVS. Note that the input point sets have all been presented in Chapter 5, and consequently, we refer the reader to the Figures 5.14 (page 166), 5.15 (page 167) and 5.17 (page 169) in that chapter for their observation. Recall from the previous chapter that, due to the clear visibility of the first dataset, Shallow Water, the resulting point set does not present high levels of noise or many outliers. Consequently, the results shown in Figure 6.20 present a smooth representation of the object. Also, due to the above mentioned larger coverage of plane sweeping methods, the surface recovered is more complete than in the PMVS case.

In the case of the Coral Reef and Tour Eiffel datasets, the surface presents some non-manifold structures. This is due to the double contours present in the original datasets.

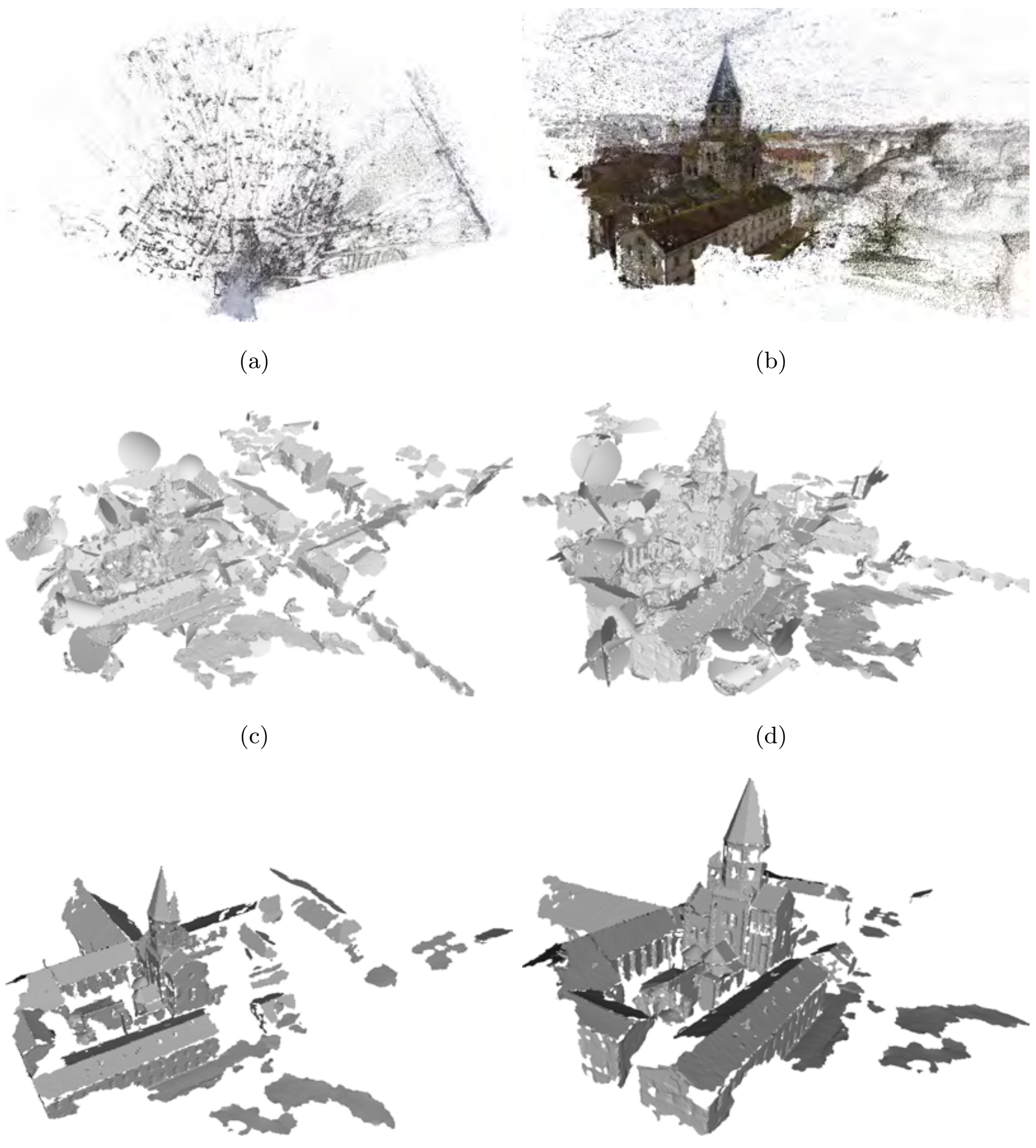
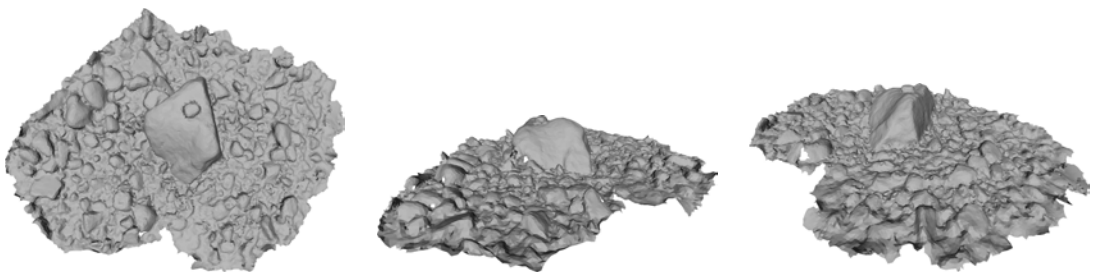
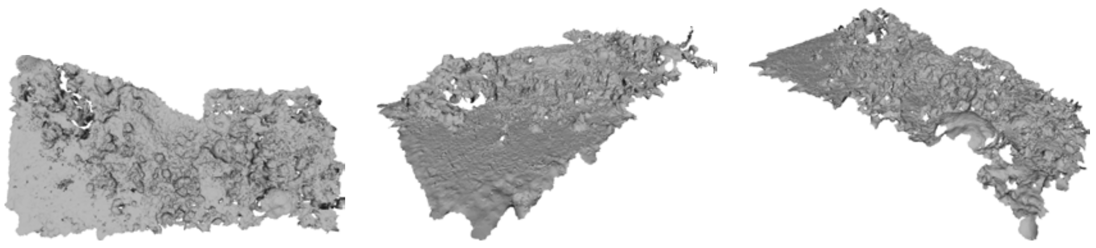


Figure 6.19: Cluny dataset. (a) Top view of input point set: 987,190 colored points with noise and many structured outliers, obtained through aerial photography and dense photogrammetry. (b) Slanted view of input point set. The high anisotropy in sampling density is mainly caused by the camera's position. Buildings closer to the camera, such as the church, are densely sampled, while the houses in the rear part are more sparsely sampled. (c) and (d); reconstructed surfaces with a small value for parameter δ_m ($k = 50$, $\delta_m = 25$). (e) and (f); reconstructed surfaces with a larger value for parameter δ_m ($k = 50$, $\delta_m = 40$).


(a) Shallow Water


(b) Coral Reef

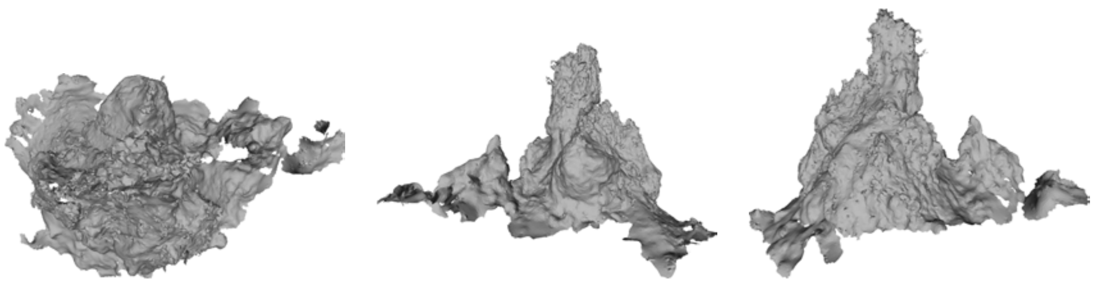
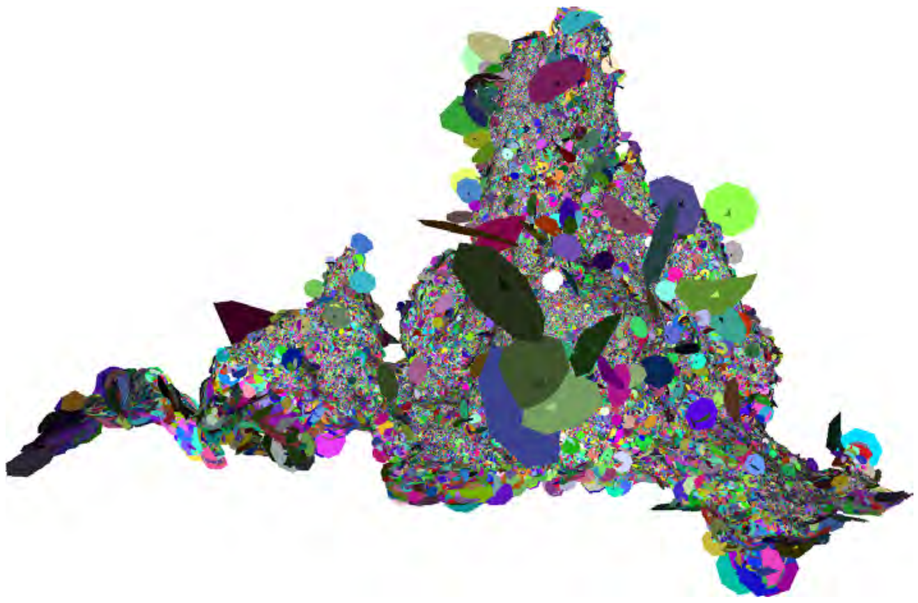

(c) Tour Eiffel

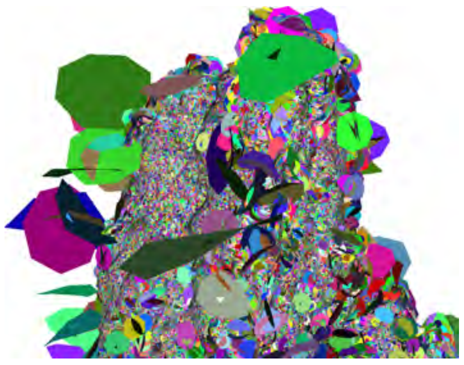
Figure 6.20: The results of our method when applied to underwater datasets whose point sets have been obtained through plane sweeping (point sets in Figures 5.14, 5.15 and 5.17, on pages 166, 167 and 169 respectively).

Note in Figure 6.21 how the splat representation in the case of outliers close to the inlier data is far from perfect. Near the surface, and specially in those places where there is higher curvature, outliers near the surface get captured by splats, which makes them larger and not coherent with their neighbors. Fortunately, the robust intersection detection eliminates most of the aberrations that these erroneous splats introduce. This is thanks to the redundancy check, so that at least two splats must contribute to an intersection test to consider it valid. Note that this is also accomplished in the parts where two splats intersect, which explains some remaining off-surfaces visible in Figure 6.21 (b).

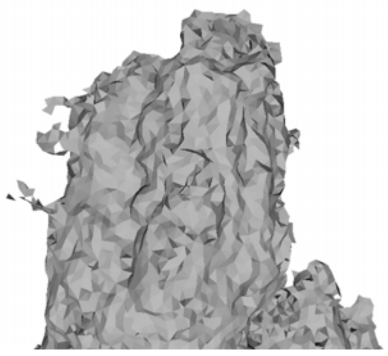
Unfortunately, the dataset of La Lune, presented in the previous chapter, could not be recovered using our methodology. The low quality of the images led to a wrong registration of the cameras, which in turn led to the creation of lots of double contours, which add to the great number of outliers and the inherent noise of the dataset. In our current frame-



(a)



(b)



(c)

Figure 6.21: Splat representation (a), and close-up of the top part of the Tour Eiffel dataset, showing the splat representation (b) and the recovered surface (c). Note how the surface is almost completely recovered even with the erroneous splats generated by the outliers near high curvature areas.

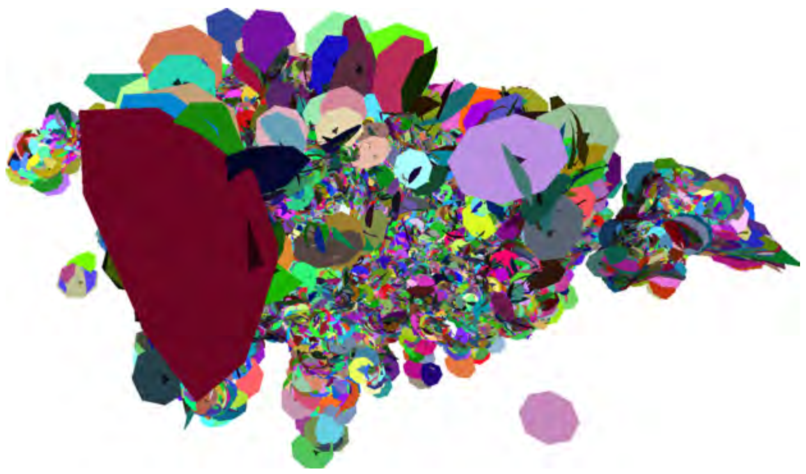


Figure 6.22: Best splat representation extracted from La Lune dataset. The self-intersection between splats prevents our method from working on this dataset. Note however that, when comparing this with the original point set in Figure 5.21 (page 172), a lot of gross outliers have been eliminated using our method.

work, in order to alleviate double contours, the only solution is to loose the δ_r threshold. With a permissive δ_r, the double contours are treated as noise, and thus generated splats merge the information on multiple contours and they consequently remove the double contour problem. However, in addition to double contours, we have a lot of outliers. This introduces the problem that, if we increase δ_r, we also loose the robustness to outliers, and thus the outliers close to the surface are not eliminated. This produces a splat representation where the splats do not fit one another, with multiple self-intersections. As previously noted, if there is no coherence between neighboring surfaces, our robust intersection detection fails. To illustrate this problem, we show the best splat representation achieved in Figure 6.22, where you can see that most of the splats self-intersect. However, it is worth noticing that gross outliers, those further from the true surface, are eliminated using our procedure. Note that the problem of self-intersecting splats is also pronounced in areas of high curvature. In these areas, if we have a sufficiently large query segment, the intersection computation can merge the contributions of various splats into a single one. However, given a smaller query segment resulting of requiring a small α_d, the intersection test may fail, as it may not intersect any splat, or only with one, thus making our robust intersection test fail. Figure 6.23 shows this phenomenon with the Shallow Water dataset, where we can see that inconsistencies arise in high curvature areas. Thus, it is obvious that in some cases we have a limitation on the smallest distance threshold α_d that we can require.

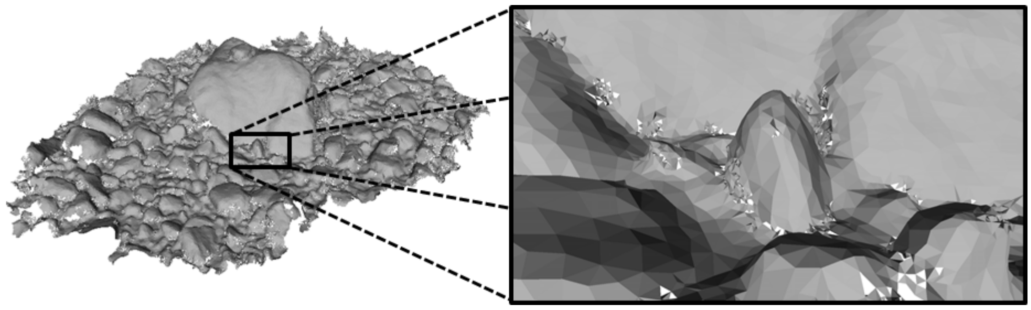


Figure 6.23: Problems of the method when the query segment becomes small enough to not merge contributions of more than a single splat. Since our robust intersection detection requires redundancy, i.e., more than a single one, to consider an intersection as valid, the intersection becomes invalid when the query segment becomes too small to intersect more than one splat. Note how this happens in areas of high curvature, where splats have less coherence between one another. Nevertheless, a more permissive δ_d results in fewer problems in these areas, as shown in Figures 6.15 (b) and 6.20 (a).

Table 6.3: Values of the parameters and running times used in each of the presented figures, r stands for row and c stands for column. Values in rows δ_r and α_r/α_d multiply the BBD of the input point set. All computations were performed on an Intel Core i7-2600 CPU @ 3.40GHz, 16Gb RAM. Running times are in seconds. Note that Figure 6.8 does not have running times since the parameters shown were used on all the corrupted sphere datasets.

				Splats				
Figure	6.4	6.6(a)	6.6(b)	6.6(c)	6.6(d)	6.7r2c1	6.7r2c2	6.7r2c3
k	100	50	50	50	50	100	100	100
δ_r (BBD)	0.005	0.01	0.01	0.01	0.01	0.005	0.005	0.005
δ_m	50	15	15	15	15	75	75	75
Run time	412.92	43.96	76.79	164.75	599.70	3919.69	9684.95	11695.6
Figure	6.7r2c4	6.7r3c1	6.7r3c2	6.7r3c3	6.7r3c4	6.8	6.9	6.10(b)
k	100	100	100	100	100	100	25	100
δ_r (BBD)	0.005	0.0025	0.0025	0.0025	0.0025	0.015	0.005	0.005
δ_m	75	75	75	75	75	50	15	50
Run time	15339.2	4975.37	10637	15369.8	19925.1	-	2.31	261.99
Figure	6.10(c)	6.10(d)	6.11(b)	6.11(c)	6.11(d)	6.13(b)	6.14(b)	6.15(b)
k	100	100	25	50	100	100	200	200
δ_r (BBD)	0.005	0.005	0.005	0.005	0.005	0.01	0.01	0.01
δ_m	50	50	15	25	25	15	50	100
Run time	261.99	261.99	17.46	22.68	35.22	285.54	816.77	377.435
Figure	6.16(b)	6.17(a)	6.18(c)	6.19(c)(d)	6.19(e)(f)	6.20(a)	6.20(b)	6.20(c)
k	100	50	250	50	50	100	25	100
δ_r (BBD)	0.01	0.01	0.0002	0.00003	0.00003	0.05	0.0239	0.0024
δ_m	50	15	150	25	40	25	5	90
Run time	68.922	35.05	18309.7	10083.2	9987.67	490.01	230.83	914.41
				Meshing				
Figure	6.4	6.6(a)	6.6(b)	6.6(c)	6.6(d)	6.7r2c1	6.7r2c2	6.7r2c3
α_r/α_d (BBD)	0.0025	0.003	0.003	0.003	0.003	0.003	0.003	0.003
Run time	211.19	43.83	50.03	69.84	74.15	94.49	118.42	98.527
Figure	6.7r2c4	6.7r3c1	6.7r3c2	6.7r3c3	6.7r3c4	6.8	6.9	6.10(b)
α_r/α_d (BBD)	0.003	0.003	0.003	0.003	0.003	0.028	0.001	0.02
Run time	107.94	109.74	116.35	114.04	114.72	-	13.48	15.32
Figure	6.10(c)	6.10(d)	6.11(b)	6.11(c)	6.11(d)	6.13(b)	6.14(b)	6.15(b)
α_r/α_d (BBD)	0.01	0.005	0.005	0.005	0.005	0.002	0.0013	0.0034
Run time	30.46	84.13	10.54	17.46	30.8	195.14	1973.50	342.22
Figure	6.16(b)	6.17(a)	6.18(c)	6.19(c)(d)	6.19(e)(f)	6.20(a)	6.20(b)	6.20(c)
α_r/α_d (BBD)	0.0029	0.0044	0.001	0.0002	0.0002	0.0045	0.0024	0.0024
Run time	277.69	62.32	82.66	214.63	49.09	194.283	223.78	231.26

6.6 Conclusions and Future Work

In this chapter we have presented a surface reconstruction method that is able to recover smooth representations of a surface under a large quantity of outliers and noise. The separation of the splat creation and the surface meshing steps makes the method modular, and provides re-usability of the results obtained at each intermediate step. Our method works without further information other than the raw point set, and the local nature of the individual splats makes the method recover, to some extent, the boundaries of the reconstructed shape.

Our approach has some limitations: the reconstructed surfaces are limited to being smooth and may be non-manifold as the surface meshing does not enforce generating manifold surfaces (in the case of requiring manifold results, a post processing method like that in Section 5.4 should be applied). Furthermore, our method is not able to reconstruct largely undersampled parts, resulting in holes in areas with missing data. Additionally, a uniform k parameter is clearly not sufficient when dealing with widely non-uniform point sets. Finally, we have seen that we cannot solve the problem of double contours along with the outlier problem in both are present in a dataset.

Future work points towards an automatic adaptive selection for k, and a parallelization of all the steps for modern multi-core architectures.

Chapter 7

Surface Reconstruction through Minimum Cuts in Graphs

7.1 Introduction

Taking a look back to the results discussed in Section 6.5, and despite the method having been proven useful in a wide range of scenarios, the reconstruction results still have some flaws. At first glance, we can detect two obvious problems.

First, splats may not fit completely with their neighbors. Moreover, in areas of high curvature, the splats are likely to self intersect. This method, as defined, works with smooth surfaces, but even in cases where a surface can be considered smooth, highly curved parts may distort the splat representation. Furthermore, the self intersection problem is aggravated by the naive splat sizing mechanism, which may lead to larger splats in these sharper areas. In both cases, the result is a non manifold surface. Recall that having a non-manifold surface is non-realistic and complicates further processing applied to the resulting mesh.

Second, it is difficult to remove outliers that fall too close to the surface with respect to the desired RANSAC distance threshold. The robust computation of splats has proven successful in removing the so called *gross outliers*, i.e., outliers far from the true surface, and whose distance to their nearest neighbors is large. Only in cases where the outliers are close enough to the surface with respect to the RANSAC threshold are they used to generate splats. Obviously, using these near-by outliers leads to a jagged, self-intersecting splat representation. However, if we take the outliers that fall very close to the true surface as part of the noise, the problem translates into a noise reduction issue.

In the local setup proposed so far in both Chapters 5 and 6, these problems were complex to solve, requiring a post processing of the resulting surface in order to recover from the non-manifold configurations. In this chapter, we change our proposal to a global implicit setup, from which a manifold surface can be extracted from the vicinity between two sub-volumes.

7.2 Overview and Contributions

We again base our method on the splat representation presented in the previous chapter. As stated above, gross outliers are supposed to be eliminated when constructing the splats, but this representation may still be noisy, in terms of self-intersecting splats refraining from obtaining a manifold surface. Considering the review of the state of the art presented in Chapter 4, and specially those in Section 4.5, we can see how, in the case of having the surface represented by local primitives, the common approach is to merge or blend them into a global Signed Distance Function (SDF). Merging different contributions in a global representation leads to significant noise reduction, i.e., the contributions of the possibly self-intersecting splats are merged, providing a smooth distance function.

Note, however, that in most cases the orientation of the local primitives is supposed to be known, usually derived from the assumption of known normals at input points, or other additional information like the sensor position is on hand. However, our splats are not coherently oriented through the surface they define. This prevents a direct computation for the distance function to be signed.

We could cast the problem as trying to orient the splats coherently. For this purpose, and up to the moment of writing this thesis, the approaches dealing with this subject are mainly variants of Hoppe's method [104]. Hoppe's method consists of propagating the orientation between neighboring primitives following an MST. In noise-free cases where there are small variations in curvature, Hoppe's method has proven to provide satisfactory results. However, in real-world data, the approach is likely to fail. Take for example the application of Hoppe's orientation method to our splat representation. In the first row of Figure 7.1, you can see how the unoriented splats (where inverted orientation w.r.t. the viewer is denoted with darker areas) are coherently oriented through Hoppe's method. From this consistent orientation, we can extract an SDF using a merging of primitives similar to the one that will be introduced in Section 7.3. However, the second row of Figure 7.1 shows a model where this orientation propagation mechanism fails. While a large part of the splats have been correctly oriented, the orientation flips at some point, leading to a globally non-coherent orientation, and consequently an incorrect SDF.

Despite this lack of coherence in orientation, we can still use the non-oriented splats to produce an Unsigned Distance Function (UDF). Unfortunately, we cannot extract a surface mesh from an UDF. Figure 7.2 shows a schematic representation of this phenomenon.

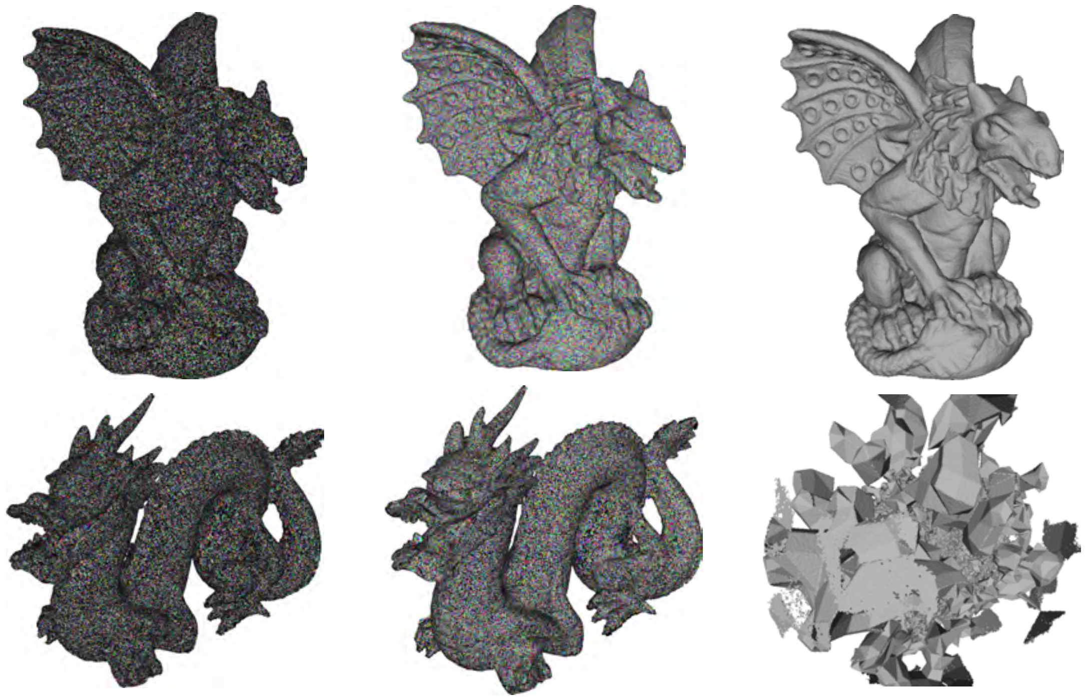


Figure 7.1: Example of Hoppe's MST orientation method applied to our splat representation. From left to right, the splats (with a randomly forced orientation), the oriented splats and the reconstructed surface. The surface reconstruction is performed in a way similar to the proposal of Hoppe [104], i.e., computing the SDF as the distance to the splat whose center is closest to the query point. The implicit distance function is evaluated at the vertices of an irregular mesh, as presented in Section 7.3, and then meshed through the RDT surface mesher.

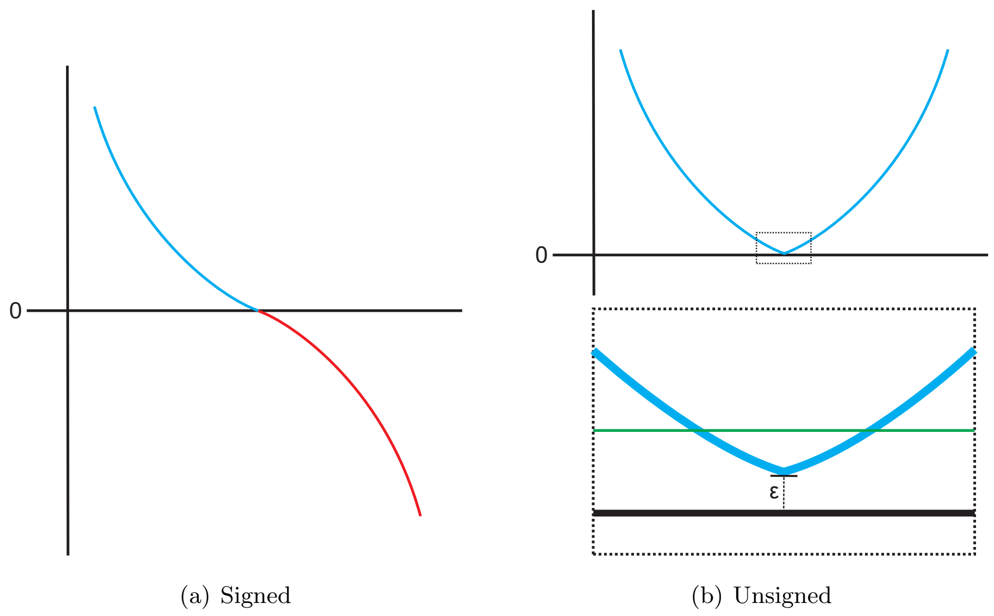


Figure 7.2: The problem of contouring a UDF. Imagine we cut a 2D slice near the surface of the signed (a) and unsigned (b) versions of the distance function defined in Equation (7.1). In the signed version (a), there is a clear zero value to isocontour located at the inflection point between positive and negative values. Whereas for the unsigned version, due to roundoff errors, an absolute zero value is never found, but only a local minima of ϵ. Note, however, that this ϵ changes for different parts in the 3D distance function, and thus cannot be fixed. If we try to isocontour the surface at a value close to zero (green line), we will obtain two valid isovalues, defining not just one surface but two, thus capturing not a surface but a *band* of volume.

Due to roundoff errors, the exact 0 value may never exist, and the surface is defined by local minima on the implicit function. Thus, if we mesh for a given isovalue close to zero, we will not obtain one 2D manifold, but two. This is so because we need the isovalue to be a clear inflexion point in the SDF. Referring again to Chapter 4, we can see that our problem falls within Section 4.5.2, i.e., we have an unsigned distance field from which we need to extract the surface. After discarding the orientation propagation in the splats, the best chance is to try to recover the sign of the SDF out of the UDF. Basically, the process consists of dividing the object into its *inside* and *outside* parts.

Note that we can refer to this subdivision as a **clustering** or a **segmentation**. In fact, the problem of inside/outside labelling is a binary segmentation problem. Inspired by the great results achieved in the computer vision community for this task, we propose using a graph formulation and use minimum cut algorithms for solving the inside/outside labelling and, consequently, the surface reconstruction problem.

We considered two approaches differing in the cutting procedure and the graph construction. However, the computation of the UDF is the same in both cases. Thus, in

Section 7.3 we start by presenting the construction of the UDF. Then, Sections 7.4 and 7.5 present two different proposals to solve the binary labelling problem. Next, Section 7.6 shows the results and discussion on the advantages and disadvantages of both methods.

7.3 Unsigned Distance Function

The purpose of building a distance function is to blend the local contributions into a global distance field. This permits the otherwise non-coherent related local surfaces to contribute to a consistent global surface represented as the zero isolevel in the distance field.

We use a variant of the implicit MLS definition [124], presented in Section 4.5.4. This method blends together the local SDF generated by the oriented points regarded as planes. We update the formula by adding support for splats of degree 2 (i.e., LBQs), and by taking into account the unsigned distance to the local surfaces instead of the signed one. The UDF at a given query point p is computed as follows:

$$u(p) = \frac{\sum_{s_i \in S} \phi_{s_i}(p) f_{s_i}(p)}{\sum_{s_i \in S} \phi_{s_i}(p)}, \tag{7.1}$$

where S is the set of splats s_i, $f_{s_i}(p)$ is the projection of p onto s_i, and ϕ_{s_i} is a Gaussian of the following form:

$$\phi_{s_i}(p) = \frac{e^{-\frac{\|p-c_i\|^2}{\sigma^2}}}{N_s}, \tag{7.2}$$

c_i being the center of s_i, and N_s the total number of splats involved in the computation. In fact, for each query point p, $u(P)$ is computed using the splats whose centers fall at a radial neighborhood of size σ. These neighborhood relationships are efficiently obtained through the use of a KD-Tree data structure. Consequently, this distance function has a bounded support, that is, it is only defined on a tubular volume around the splats, at a distance of σ at most.

Basically, this UDF is a weighted mean of the individual distance contributions of the splats falling near a given query point. It is obvious that the distance function presented does not take into account gross outliers, which are supposed to be eliminated by the splats creation procedure. Note also that we just use the centers of the splats and do not take into account their size in the distance computation. Thus, we call them splats just for coherence with the previous chapter, but in reality we use either points with normals or LBQs, corresponding to splats of $d = 1$ and $d = 2$ respectively, and unoriented in both cases.

Despite the use of a KD-Tree for a rapid query of neighborhood relationships, computing the function at an arbitrary point in space is a costly operation. In order to alleviate the computational burden of intensively querying this implicit function, which is likely to be required during the surface extraction step, we discretize our domain. At each of the vertices in the partitioned domain, we compute the UDF using Equation (7.1). Then, for an arbitrary query point in $\mathbb{R}^3$, the function value is returned using a linear interpolation of the values stored in those vertices. Moreover, instead of following the traditional approach of using a regular grid, we lean toward using an irregular tetrahedral grid that adapts to the density of the input points.

Up to this point, in this thesis we have seen the Delaunay refinement associated to the RDT meshing. In this case, we use the Delaunay refinement in order to obtain the tetrahedral grid discretizing our working space. Starting from a Delaunay triangulation containing the centers of the splats, new vertices are inserted iteratively until the given criteria are fulfilled. In this case, we force the ratio between the edge of the minimum length and the circumradius of the tetrahedra to be lower than the threshold α_{re} (also referred as the radius-edge ratio). Note that we are aiming at having a faithful approximation of the distance function near the centers of the splats (which are located near the original points in the input P). By forcing the radius-edge bound to be relatively low, the tetrahedra become larger when far from the points, giving a rough approximation, and smaller when closer to the input data, providing a more precise approximation.

As previously stated, the values of the distance function are computed at the vertices of the triangulation. Thus, for an arbitrary query point, the tetrahedra containing it is localized and the function value is interpolated using the values at the vertices of the tetrahedra along with the barycentric coordinates of the point. Obviously, if the query point is outside the band, the value is undefined. We refer to the vertices of this data structure as U, and to its refined Delaunay structure as $\text{Del}(U)$.

7.4 S-T Cut

Given the unsigned function described above, our goal is to recover its sign. That is, we want to distinguish between the part of the volume inside the object from the part outside, and then simply apply the corresponding sign to each part to convert the original UDF into a SDF. Certainly, this problem can be considered as a binary labelling problem: we want to partition the vertices U in the above mentioned tetrahedralization into the two labels *in* and *out*. Given the extensive use of the S-T cut methods in binary optimization, our first proposal is to adapt this method to perform the partition on our scenario.

7.4.1 Theoretical Bases

An S-T cut divides a set of nodes in a specific type of graph into two disjointed sets minimizing the cost associated with the removed edges. Commonly referred to in the literature with the rather confusing name of *Graph Cuts* or, alternatively, the min-cut/max-flow algorithm, this method obtains an exact minimization for binary optimization problems. This technique has been extensively used in a wide range of applications in computer vision and graphics [95, 178, 126, 127, 131, 175, 135, 132, 133, 99, 112, 107].

We start by presenting the graph terminology that will be used throughout this chapter. A graph $\mathsf{G} = \langle \mathsf{V}, \mathsf{E} \rangle$ is composed of a set of vertices V and the set of edges E joining them. In the specific case of S-T cut algorithms, the set of nodes V contains two special nodes, s (source) and t (sink), referred to as the *terminal* nodes, so that $\mathsf{V} = \mathsf{P} \cup \{\mathsf{s}, \mathsf{t}\}$, P being the rest of non-terminal nodes. Each edge joining vertices $\mathsf{v_i}$ and $\mathsf{v_j}$ stores a given weight $w(\mathsf{e}_{i,j})$. We differentiate between these edges by calling them terminal edges M, if they join a terminal vertex with a non-terminal one, or non-terminal edges N, which only describe interactions between non-terminal vertices.

An S-T cut of the presented graph is a partitioning of the graph nodes into two subsets, S and T, so that $\mathsf{s} \in \mathsf{S}$ and $\mathsf{t} \in \mathsf{T}$. The cost of cutting a graph equals the sum of weights on the severed edges $w(\mathsf{e}_{i,j})$, so that $\mathsf{v_i} \in \mathsf{S}$ and $\mathsf{v_j} \in \mathsf{T}$. Thus, the problem is then to find the minimum S-T cut from all the possible cuts in the graph. A useful result in combinatorial optimization is that the minimum S-T cut is dual to the problem of finding a *maximum flow* from source s to sink t [81].

From the binary optimization point of view, the terminal nodes s and t represent our possible labels. Suppose we can define a cost for assigning each node to a given label. Also, assume the labelling problem does not depend only on the nodes themselves, but also on the labels of their neighbors. Thus, our problem basically requires an optimization of the labels for each vertex in the graph, by enforcing spatial coherence between neighbors. This results in the minimization of an energy function composed of a data term and a smoothing term:

$$E(l) = \sum_{p \in \mathsf{P}} T_p(l_p) + \sum_{\mathsf{e}_{i,j} \in \mathsf{N}} V_{i,j}(l_i, l_j), \tag{7.3}$$

where $T_p(x)$ is the data penalty term, $V_{i,j}(x)$ is the interaction potential, and l is the binary labelling to optimize, defined as follows:

$$l_p = \begin{cases} 0 & \text{if } \mathsf{v_p} \in \mathsf{S} \\ 1 & \text{if } \mathsf{v_p} \in \mathsf{T}. \end{cases} \tag{7.4}$$

Note that, in our case, the 0 and 1 labels correspond to the *inside* and *outside* of the object to be reconstructed.

Related to the above mentioned interaction potentials, $T_p(x)$ indicates per-vertex labelling preferences, while $V_{i,j}(x)$ encourages spatial coherence and penalizes discontinuities between neighboring labels. Kolmogorov and Zabin [128] proved that a globally optimal labelling for the energy in Equation (7.3) can be found using the minimum cut on an S-T graph, as long as the following regularity condition applies:

$$V_{i,j}(0,1) + V_{i,j}(1,1) \leq V_{i,j}(0,1) + V_{i,j}(1,0). \tag{7.5}$$

In our case, presented in the following section, all the edges have undirected weights, but the general method is described for the case of directed weights. As suggested in many other approaches [107, 132, 133, 99], we use Boykov and Kolmogorov algorithm [37] to solve for the minimum cut.

7.4.2 Our Approach

We define our graph G using the same connectivity of Del(U), that is, the vertices P of the graph correspond to U, and the edges in the adaptive structure containing the UDF also define the relationships between vertices in G. Additionally, some of the vertices in the graph have a pair of edges joining them to both the s and t nodes. This leads to the graph definition depicted in Figures 7.3 (b,c) for the 2D case.

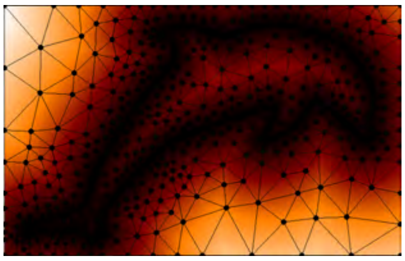
In order to apply the above defined optimization method to the U nodes in our space partition, we need two main ingredients. On the one hand, we have to infer a confidence for each $p_i \in U$ to be inside or outside the shape. On the other, we need to devise an inter-nodes weighting. Equivalently, we need to define both terminal and smooth weights.

In the case of the smooth weights, these are derived directly from the unsigned distance function values. Thus, each $e_{i,j} \in \mathsf{N}$ has a weight $w_s(e_{i,j})$ which corresponds to the direct evaluation of the unsigned distance function along the edge. Note that, since our vertices in the graph coincide with those in Del(U), the UDF along the edge is constant, and thus the weight simplifies to:

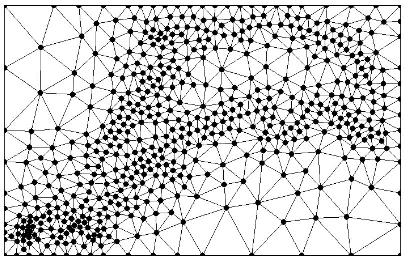
$$w_n(e_{i,j}) = \left(\frac{u(p_i) + u(p_j)}{2}\right)^{\beta}, \tag{7.6}$$

which is basically the mean value of the function value at the endpoints of the edge, and where the power β is a user parameter allowing the emphasis of the minimum of the distance function as suggested in Hornung and Kobbelt [107, 106]. Using this definition for the smooth weights enforces the minimum cut to pass through the minimum of the unsigned distance function. Note that our distance function definition is defined just on a narrow *sigma*-band near the centers of the splats. Thus, smooth weights $w_n(e_{i,j})$ are only defined inside this band. For the rest of the edges, a default constant value is added, enforcing the propagation of labels from the neighbors in the band.

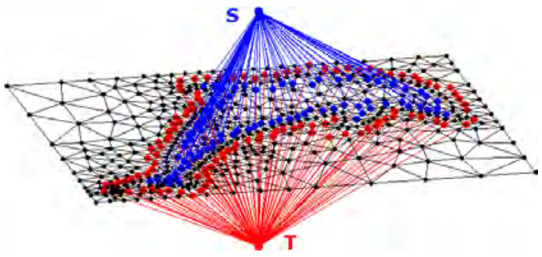
For the terminal weights, we take advantage of our splat representation. Recall that this representation is already a good approximation of the surface of the object, and that we have defined a robust intersection method using RANSAC in Section 6.4.2. Thus,



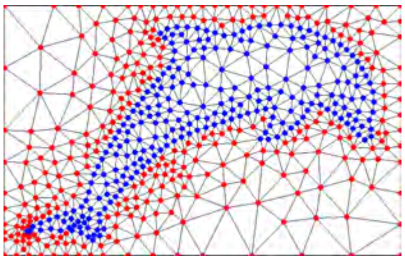
(a) UDF



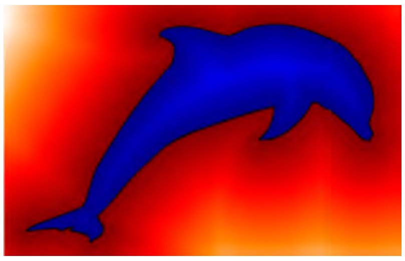
(b) Graph (smooth weights)



(c) Graph (terminal weights)



(d) Optimized



(e) SDF

Figure 7.3: 2D depiction of the pipeline followed in this chapter. Starting from an UDF evaluated in an adaptive grid (a), a graph structure is built using the connectivity defined by the triangulation (b). Additionally, for the case of S-T cut, some of the vertices in the graph are connected to two special nodes; `s` and `t` (c). After the graph optimization, we obtain a binary partition of the vertices in the graph (d), which is used to sign the original UDF (e). Using the SDF, we can extract the surface, the curve in this case, at its zero level set.

we use a procedure similar to that presented in Mullen et al. [159], but adapted to our representation. Basically, we induce the confidence of a point as being inside or outside the object by throwing random rays originating from that point in multiple directions, and counting the number of intersections with the splat representation.

An intuitive approach to knowing whether an arbitrary point in space is inside or outside an object is to count the number of intersections between a ray with its origin at this point and the surface of the object. When the number of intersections is *odd*, the point is inside the object, and when *even*, outside.

Since we do not know the surface of the object, we substitute it with the splat representation. For each vertex/point p in the triangulation, we throw a given number of rays, N_{rays}, with a starting point at p in random directions, and count the number of intersections. Given all these intersections, we count those resulting in an even number, i_{even}, and those giving an odd number, i_{odd}. Note that we do not throw a single ray, taking into account that the ray-splats intersection query may fail in some cases. Using all these tests, the confidence for a given point to be part of the inside or the outside, and consequently its weight with the s and t nodes is:

$$\begin{aligned} w_s &= i_{even}/N_{rays} \\ w_t &= i_{odd}/N_{rays}. \end{aligned} \tag{7.7}$$

Note that here, without loss of generality, we have fixed the s node, and consequently the S set of the cut, to represent the inside of the object, while t (resp. T) represents the outside.

Note, however, that the robust ray-splats intersection query has been proven to work on large scales, i.e., far from the splats, but to be not so reliable with small scales, i.e., near the splats. Consequently, the vertices in U outside the σ-band are more suitable for reliable stochastic ray signing as presented above. Furthermore, despite the fact that the ray-splats intersection query is optimized using Axis-Aligned Bounding Boxes (AABB) trees, its repeated use could lead to a drop in time performance for the method. In order to alleviate computational complexity, we simply apply the stochastic ray confidence computation to the points at the interphase of the σ-band. That is, we just compute the confidence for a vertex if at least one of its adjacent neighbors is inside the band (see Figure 7.3 (c)). This means that only a small subset of vertices has a terminal weight and, for the rest of the vertices, no terminal edge is added. Thus, points with no terminal weights obtain the final label due to the propagation ruled by the smooth weights.

In order to get a visual depiction of the pipeline described here, refer to Figure 7.3.

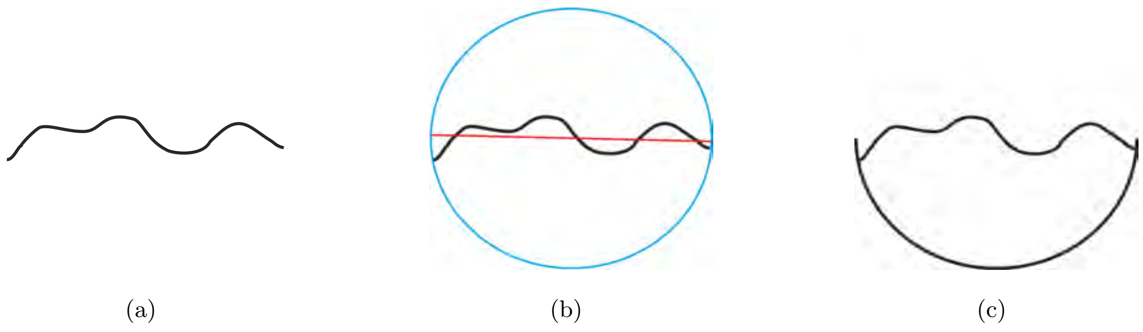


Figure 7.4: 2D Schematic of the extension for the S-T cut method to handle bounded surfaces. The curve in (a) is not closed, and consequently it does not define an inside or an outside. We compute a global frame using PCA, and use it to chop the surface into two spherical caps, as depicted in (b). One of the spherical caps is used to *virtually close* the surface as presented in (c), so that we can disambiguate the inside/outside computation.

7.4.3 Extension to Open Surfaces

It is obvious that the above presented S-T cut approach has a clear flaw in the scenario of this thesis. We have seen that the majority of objects retrieved from the seabed describe bounded surfaces. In these cases, there is no *inside* notion, and consequently the inside/outside confidence procedure as previously presented does not work. We adapt our method to work in this scenario by *simulating* the surface to be closed.

We start by computing a global plane using PCA with the centers of the splats. Then, we compute the bounding sphere containing these centers and use the previously computed plane to chop it off into two spherical caps. This way, one of the caps is used as the reference to virtually define an inside/outside part of the shape. In order to do so, during the inside/outside labelling through stochastic ray throwing, we count any intersection with the selected spherical cap as a valid intersection. This process is intuitively depicted in the schematic in Figure 7.4.

Note that we do not distinguish which part is which, i.e., we randomly label one of the two spherical caps resulting from stabbing the bounding sphere by the global plane as inside or outside. Since just the surface is required, the retrieved solution is valid up to a possible global orientation change of the resulting triangle mesh after the surface extraction step. Nevertheless, due to the insertion of this virtual spherical cap into the system, the retrieved surface is closed. Thus, we create some parts of the surface that are obviously not part of the real bounded surface. These parts of the surface should be eliminated, using the approach that will be presented for the *Normalized cut* case in subsequent sections.

7.5 Normalized Cut

We have seen that the S-T cut technique requires providing a notion of some of the points to be inside or outside the shape. This procedure is necessary with several state-of-the-art methods which also try to find the optimal separation of inside/outside volumes. However, in our specific case, this knowledge is just a byproduct, since the final goal is to recover the interface between the two volumes, i.e., the surface of the object. Given that we are just interested in this separating surface, in this section we propose a method to partition the volume into two, but disregarding which part is the inside or the outside. For this purpose, we use another commonly referred to graph-based technique applied in solving the binary segmentation problem: the *Normalized cut*.

Spectral methods are extensively used in clustering and segmentation in image processing [193, 151, 56, 75, 121, 150]. It is also worth noticing that, even if originally described using graph theory, Normalized cuts have a direct formulation as a *Random walk* [149], or as a separation using hyperplanes (similar to SVM) [173], among other interpretations.

It is also worth noticing that the Normalized cut has been used with minor changes in surface reconstruction by Kolluri et al. [125]. Note, however, that their method is interpolation-based, and thus its definition is completely different from ours. While they were concerned with the outlier rejection problem, we are more concerned with the attenuation of noise. That is, in their approach, the spectral cut was used to disregard outliers, while, in our case, we use it to partition a volume from an implicitly defined UDF.

7.5.1 Theoretical Bases

All in all, the question we want to answer is: can we solve the partitioning problem without having to rely on a specific labelling? Taking a look back at Equation (7.3), this results in removing the data term, leaving the energy to minimize as follows:

$$E(l) = \sum_{\mathsf{e}_{i,j} \in \mathsf{N}} V_{i,j}(l_i, l_j). \tag{7.8}$$

Thus, we have to deal with the general definition of a minimum cut: the graph has to be partitioned into two groups , regardless of their label, so that the edges of the same group have a high weight, and, on the contrary, edges between different groups have low weights. Since they do not have a specific meaning in this case, we rename our binary regions as A and B, so that the cut can be defined as follows:

$$\mathrm{cut}(A, B) = \sum_{i \in A, j \in B} w(\mathsf{e}_{i,j}), \tag{7.9}$$

which basically means that the minimum cut corresponds to the one minimizing the total weight of the edges removed.

However, as pointed out by Wu et al. [208], minimizing the cut directly, as described in Equation (7.9), favours the cut of a small set of edges, i.e., severing a few edges often leads to a minimization of the cut. Conceptually, we want the groups in this partition to be relatively large with respect to the total number of nodes. The normalized cut method tries to overcome this problem by forcing the sum of weights in both parts to be *similar*. This is obtained by normalizing the cost of the cut relative to the cost of all the edges in each region:

$$NCut(A,B) = \frac{\mathrm{cut}(A,B)}{cost(A)} + \frac{\mathrm{cut}(A,B)}{cost(B)}, \tag{7.10}$$

where $cost(X)$ is a sum of the weights of the edges contained in the set X. This definition promotes the two sets A and B to be larger and of similar cost.

As pointed out in the original article by Shi and Malik [193], solving this problem is NP-hard. However, if we change the labelling from pure binary to continuous, the problem can be reformulated into a minimization that can be solved exactly.

A key ingredient for the minimization is the Laplacian matrix L, containing at $L(i,j)$ the similarity measure between $\mathsf{v_i}$ and $\mathsf{v_j}$. Also, the matrix D, commonly referred to in graph theory as the *degree* matrix, having the same size as L and containing on its diagonal elements d_i the sum of all weights for a given vertex (i.e., the sum of weights of its associated edges).

Let us define our labelling to be contained in the vector l:

$$l_i = \begin{cases} 1 & \text{if } \mathsf{v_i} \in A \\ -1 & \text{if } \mathsf{v_i} \in B. \end{cases} \tag{7.11}$$

Then, we define the following ratio for a given labelling:

$$b = \frac{\sum_{l_i>0} d_i}{\sum_{l_i<0} d_i}. \tag{7.12}$$

Using the b value, a slightly different version of l can be defined as $m = (1+l)-b(1-l)$. Note that this change only reduces the labelling to be reformulated as:

$$m_i = \begin{cases} 2 & \text{if } \mathsf{v_i} \in A \\ -2b & \text{if } \mathsf{v_i} \in B, \end{cases} \tag{7.13}$$

which is equivalent to the previous one (Equation 7.11). However, by using this new labelling definition, we can cast the minimization of Equation (7.10) as follows (for a throughout derivation, the reader is referred to the original paper [193]):

$$\min_l NCut(L) = \min_m \frac{m^T(D-L)m}{m^T D m}. \tag{7.14}$$

Since the m labelling is still binary, the problem remains NP-hard. However, if we relax the problem by allowing the solution to be continuous, the minimization in Equation (7.14) can be converted to a Rayleigh quotient. Using $z = D^{\frac{1}{2}}m$, the problem becomes:

$$\min_{m} \frac{z^T D^{-\frac{1}{2}}(D-L)D^{-\frac{1}{2}}z}{z^T z}, \tag{7.15}$$

which is, indeed, a Rayleigh quotient, and can be minimized by z corresponding to the smallest eigenvector satisfying:

$$D^{-\frac{1}{2}}(D-L)D^{-\frac{1}{2}}z = \lambda z. \tag{7.16}$$

Since the Laplacian matrix is positive semidefinite, its smallest eigenvalue is zero, with an associated eigenvector whose components are all 1. Thus, we do not use the first, but the second smallest eigenvalue to minimize Equation (7.16). Therefore, the solution is the second smallest eigenvector λ_2 of $D^{-\frac{1}{2}}(D-L)D^{-\frac{1}{2}}$.

Since this solution gives continuous values, they have to be discretized somehow into the desired binary labelling l. This is easily achieved by looking at the sign of each value:

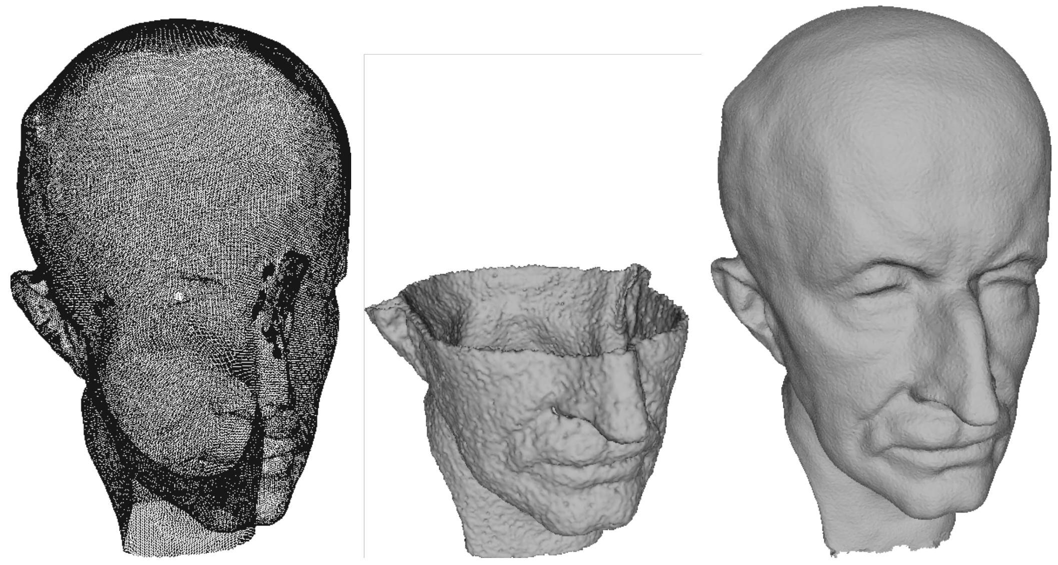
$$l_i = \begin{cases} 1 & \text{if } \lambda_{2,i} \geq 0 \\ -1 & \text{if } \lambda_{2,i} < 0. \end{cases} \tag{7.17}$$

7.5.2 Our Approach

Note that the underlaying graphs of the two techniques proposed in this chapter are very similar, just the links to S-T sites are missing in the Normalized cuts case. Thus, the filling of the graph and the creation of the weights is exactly the same. In this case, since there is no additional labelling hint, the parameter δ, used to stress the unsigned distance function minimum, has more relevance than in the case of the S-T cut.

Note, however, that the small σ-band used in the S-T cut case may not be sufficient for the present case. If this band is too small, the cut criterion could lead to problems like the one posed in Figure 7.5. Since the σ band is too narrow, the cut that minimizes the cost, and also balances the sum of edge weights in each partition, separates the edges into two parts that do not necessarily pass through the minimum of $u(x)$. In order to force the cut to pass through the minimum of the UDF, we need a larger σ band, so that the weights on each side of the band take greater importance in the balancing factor (i.e., the term $cost(X)$ in Equation 7.10).

Unfortunately, increasing the σ value leads to an increase of the computational cost. For this reason, we propose constructing a secondary band governed by the parameter σ_2. In this σ_2-band, we compute a coarser approximation of the function defined in Equation (7.1) by not taking into account the full radial neighborhood around the query point, but



(a) Point set (b) Smaller band result (c) Larger band result

Figure 7.5: Relevance of the σ band size for the Normalized Cut method. When applied to a narrower σ band (b), the Normalized cut may be minimized by a cut that does not pass through the minimum of the UDF. By enlarging the σ band (c), the weight of the edges on each side of the partition increases, forcing the cut to pass through the minimum in UDF.

only the first k_σ-nearest neighbors. This reduces the computational effort and provides an acceptable approximation. Note that having a coarser approximation of the UDF when far from the input points is not important in our case, since we are interested in the minimum of the function, and this is located inside the finer σ-band. We only use this secondary band to increase the importance of $cost(X)$ for each part of the partition, in this way, promoting the cut to pass nearer the minimum of our UDF. It is also worth mentioning that it may be interesting in some cases to use this low-quality version of the distance function in the S-T cuts case, for instance when a small σ results in holes appearing on the surface and we want our global function to fill them smoothly. Obviously, the larger the band we consider, the larger the global implicit function support, and consequently the larger the reconstructed surface area.

Also, note that the Normalized cut procedure described in Section 7.5.1 assumes the Laplacian to contain a single connected component. In cases where more than one single object is represented in the scene, and given that the graph is only defined inside a limited extent of the σ_2-band, the graph may contain more than one single connected component. Thus, once the graph Laplacian matrix is built, we separate it into connected components, and apply the method to the largest set. Notice that by just applying it to the largest one, we assume that there is only one object in the scene, and that other connected components are formed by some remaining outlier splats. When allowing for multiple objects in a scene, the cut should be computed on each connected component, to obtain a separate surface for each of them. Finally, in order to compute the required eigenvalues and eigenvectors, we exploit the high sparsity of our Laplacian by using the Lanczos algorithm [91].

After obtaining the cut, we flip the sign of one of the regions, A or B (it does not matter which), and extract the surface using a surface mesher on the resulting SDF. Finally, the results presented in the Results section demonstrate that the differences between the S-T cut and the Normalized cut methods are not that great. What is important is to emphasize that, in the second case, we do not use labelling hints at all. Thus, we prove that solving the surface reconstruction by partitioning the working volume does not require the knowledge of a specific labeling, but just the partition itself.

7.5.3 Removing Hallucinated Triangles

Note that the partition presented above takes place only in the σ band. Thus, we give a sign to a slab of volume, and consequently, the retrieved surface mesh does not contain only the part we are interested in, i.e., near the input splats, but also some hallucinated triangles corresponding to the closing of this volume (see Figure 7.6 (d)). Following the nomenclature proposed by Jancosek and Pajdla [113], we refer to *hallucinated triangles* as those which are part of the reconstructed surface obtained so far, but that do not correspond to the real surface (i.e., they are far from the input splats in this case). In order to remove these artifacts, we eliminate the hallucinated triangles by using the $u(x)$ value of their vertices in the original UDF. An example of the reconstructed surface before and after removing hallucinated triangles is presented in Figure 7.6.

Obviously, we have a large number of vertices having a $u(x)$ close to zero, corresponding to the part of the mesh close to the surface, and another large part having $u(x)$ values that progressively increase the farther away from zero they are, corresponding to the hallucinated part. Basically, we need to infer if a vertex is part of the surface by checking if its $u(x)$ is close enough to zero. However, this notion of *close* is not defined for a given dataset. For this purpose, we use the MSSE method to detect such a gap between $u(x)$ values close to zero and the rest. We compute the scale d of the variance for the values close to zero, i.e., we compute the MSSE procedure starting from the sorted $u(x)$ value located at a very low quantile, fixed to $o = 0.1$ in all the presented cases. Then, we simply remove the triangles having a vertex whose $u(x) > 2.5d$ from the surface mesh. Note that, as previously mentioned, we also apply this method when using the S-T cut method in the case of bounded surfaces. Another possible solution to this problem not explored in this thesis would be to modify the isosurface extraction method directly to work just in the parts where the UDF is below a threshold, however, automatically setting this threshold remains an open issue.

7.6 Results

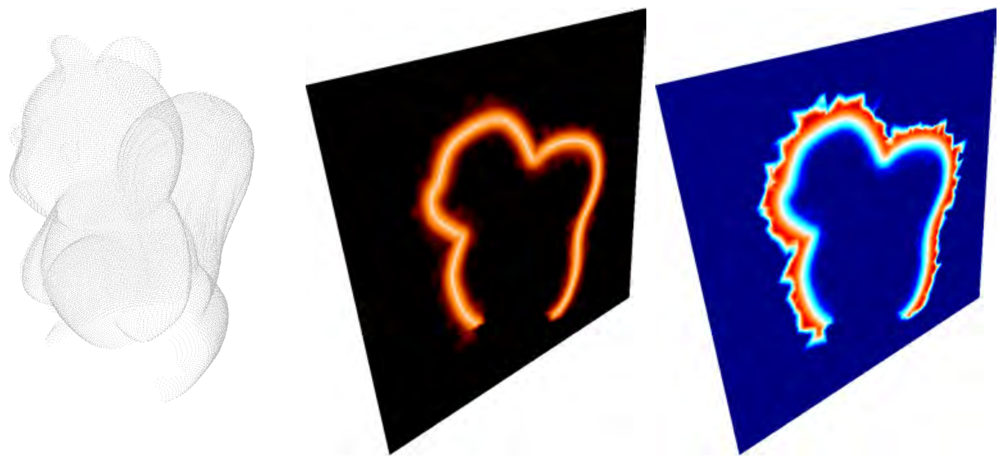
In this section we exhibit the results obtained using the two graph cutting techniques presented above. We list the parameters used to obtain the presented results for each method in Tables 7.1 and 7.2, along with the running times required for each experiment. It is worth noticing that the σ and σ_2 parameters are computed with respect to the average spacing between the input centers of the splats, calculated using the mean distance to the 6-nearest neighbors. The present section follows the structure presented in previous chapters: we start by validating both methods on range scan data to then focus on their application in a multiple view stereo scenario, and we finalize by reviewing their application in underwater datasets.

A relevant property to notice is that, since we are contouring a generalized SDF, we drop the dependency that the methods in Chapters 5 and 6 had of using the RDT surface mesher. Instead, we can use any contouring algorithm from those presented in Section 3.4.1, like for instance, the widely used marching cubes as proved in Figure 7.7 (a). However, given the aforementioned advantages of RDT meshing in terms of the quality of triangles (observable in Figure 7.7 (b)), we use this for all the results contained in this section.

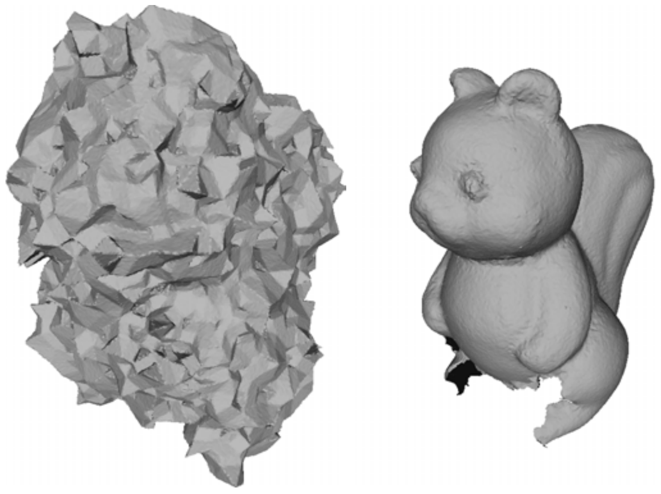
Additionally, when compared to the method presented in the previous chapter, we are able to obtain manifold surfaces, amenable to further post processing. Take, for example, the complex Stanford Dragon dataset, presented in Chapter 6, and note that the differences between the results obtained by both the S-T and Normalized cut algorithms are imperceptible. Further examples validating the application of both methods to several range scans datasets can be seen in Figure 7.9.

As pointed out in Section 7.3, we also open the door to the use of directed points (i.e., points with unoriented normals) in our framework. Many proposals in the literature deal with the problem of finding robust but unoriented normals [104, 10, 158, 68, 8, 139]. In each of these cases, the problem is what to do with this directed but unoriented point set in order to obtain the surface. Using our Normalized cut for this purpose, we can retrieve a manifold surface. For the case of the S-T cuts method, we would require splats of a limited extension to use this approach. This reduces to using the linear approximation of the directed point as local surface, and compute its extent, for instance, as explained in Section 6.3.3. An example of of the algorithms' behaviour when used with linear primitives can be seen in Figure 7.10.

Regarding range scanning applied underwater, it is well known that acoustic range



(a) Point Set (b) UDF Slice (c) SDF Slice



(d) Hallucinated Triangles (e) After Filtering

Figure 7.6: The effect of hallucinated triangles on the Normalized Cut method. Note how the reconstruction in (d) is covered by the meshed part of one of the volume slabs that has been meshed. This is due to our procedure giving a coherent sign only inside the σ_2 band. This phenomenon can be seen in (b) and (c), showing a 2D slice of the unsigned and signed versions of the 3D distance function respectively. After the automatic hallucinated triangle removal, we can see the underlying surface in (e).

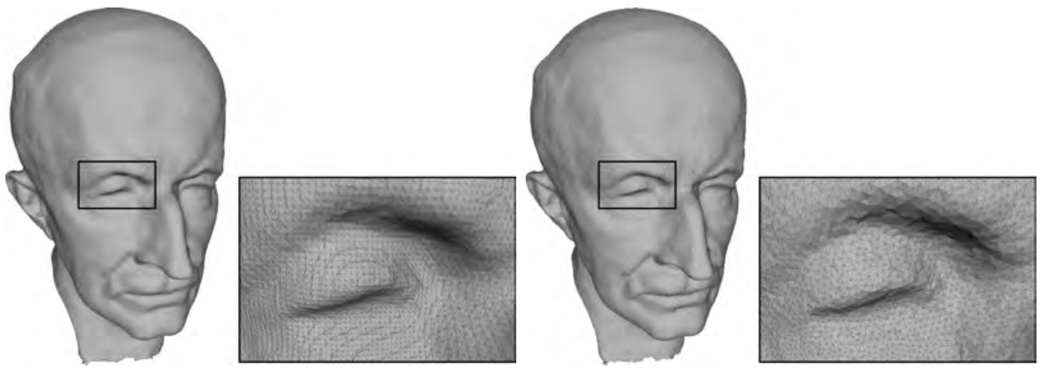

(a) Marching cubes (b) RDT surface mesher

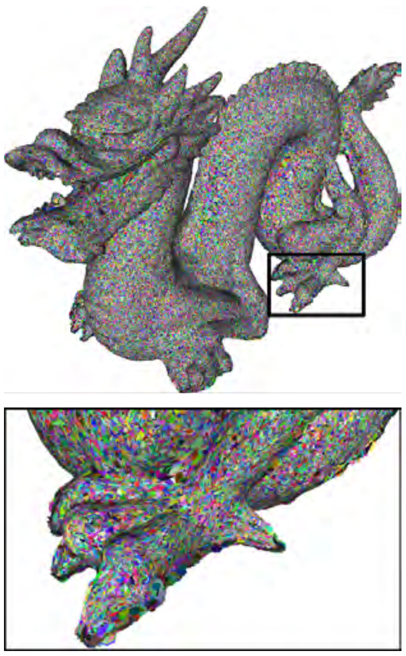
Figure 7.7: Comparison between Marching cubes (a) and RDT surface mesher (b) for the extraction of a surface. Note in (a) how the number of triangles is larger and their shape highly non-uniform, while they are better shaped (i.e., closer to regular) when using the RDT surface mesher. In both cases, the SDF used to extract the surface was constructed using the Normalized Cut method.

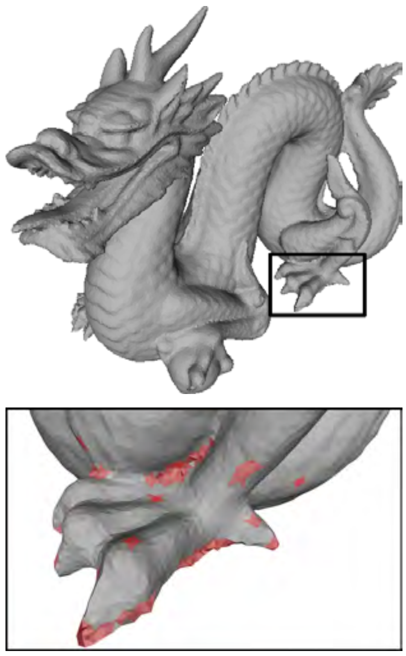
sensing is the preferred tool for mapping large underwater areas, thanks to their long working distance. For this reason, we test our methods against some datasets of this modality.

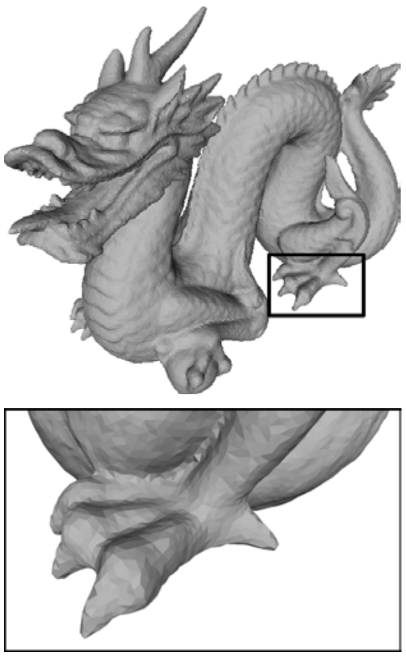
Problems like reflections and the inherent low resolution of acoustic sensors result in the point cloud retrieved using this technology being quite noisy. Additionally, the poor navigation readings used to align all the range scans into a single reference frame causes several double contours in unprocessed point sets. As pointed out in Section 6.5, having outliers and double contours generally creates a noisy splat representation with self-intersecting splats not amenable to our robust segment-splats intersection procedure. Thus, in these cases the S-T cut method, which uses this feature, cannot be used. For this reason, we only test acoustic range datasets using the Normalized cut method.

The first acoustic dataset presents a point set obtained using a multibeam sonar scanning a small underwater Mound rising from a 40 to 27 m depth located near the harbour of Sant Feliu de Guíxols, on the Costa Brava of Catalonia, Spain (see Figure 7.11). With the sonar located in a slanted orientation towards the object, the point set was automatically obtained with an adaptive replanning strategy that uses stochastic trajectory optimization to reshape the nominal path to cope with the actual target structure perceived in real time during the exploration [86]. As can be observed in Figure 7.11, despite the outliers and double contours present in the final data retrieved, the Normalized cut is able to obtain a consistent surface, although some parts of the surface are not reconstructed.

The second acoustic dataset corresponds to a profiling sonar survey of the interior of an underwater cave located in L'Escala, also on the Costa Brava. During the survey,


(a) Splats


(b) Splats Mesher


(c) S-T Cut

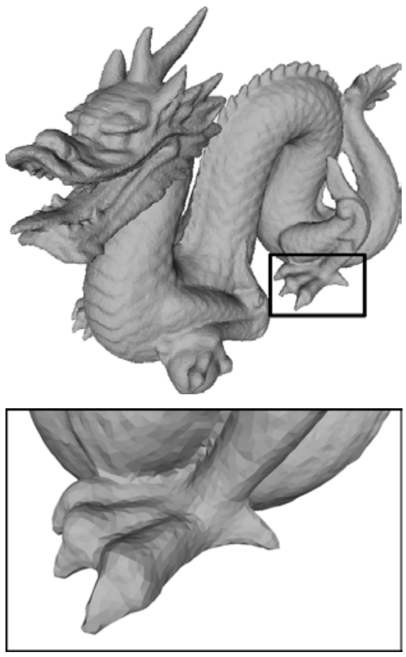
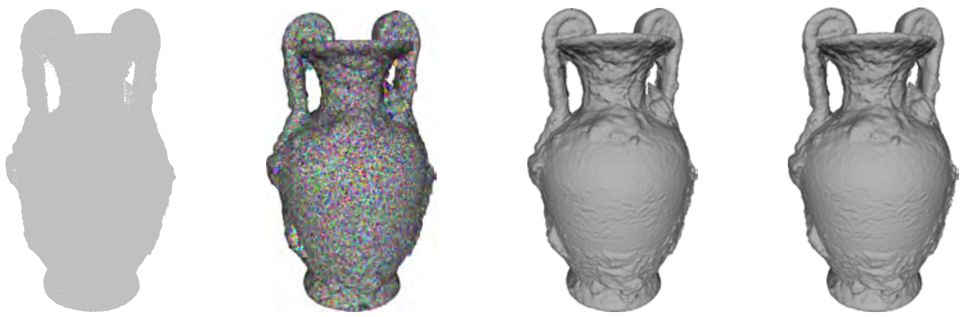
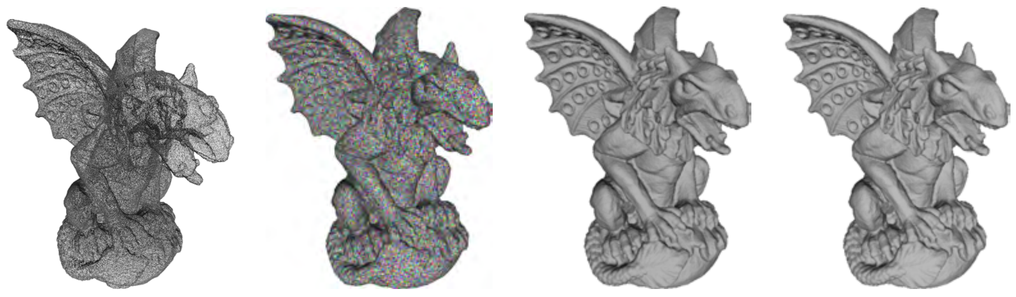
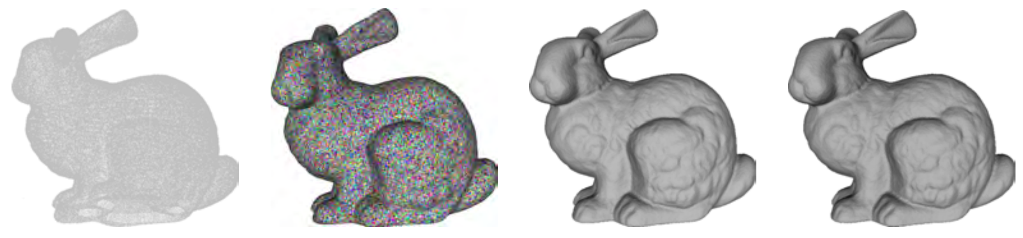

(d) Normalized Cut

Figure 7.8: Using the same splat representation shown in (a), we present a comparison of the splats mesher (b) and the two methods proposed in this chapter – (b) for S-T cut, and (c) for Normalized cut. Note that in the lower part of (b), we have emphasized the triangles containing some non-manifold edges in red.


(a) Amphora (from Aim@Shape)


(b) Gargoyle


(c) Stanford Bunny

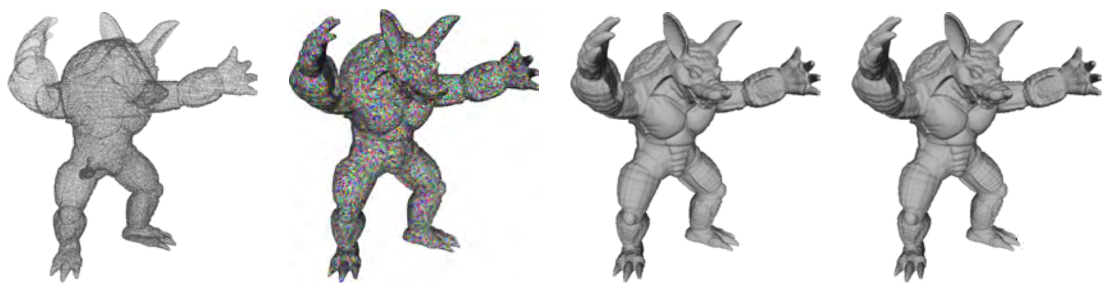
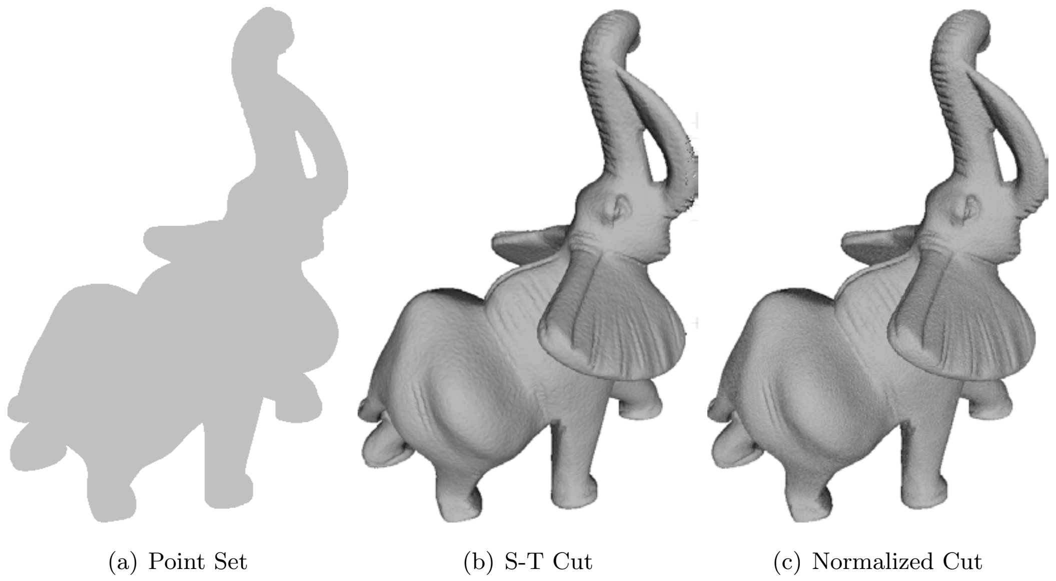
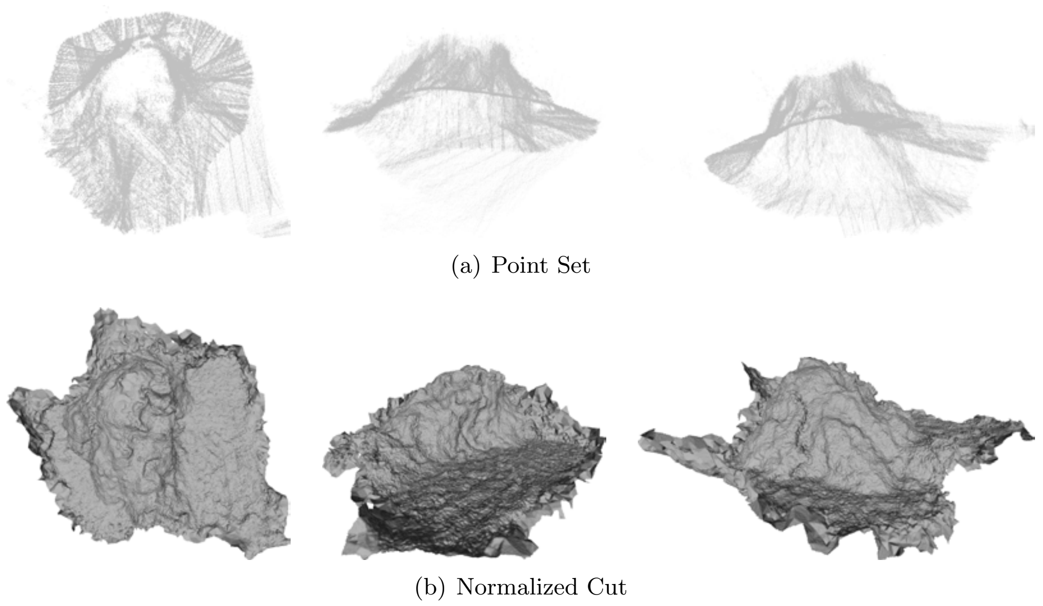

(d) Stanford Armadillo

Figure 7.9: Examples of the methods' behaviour when applied to range scan datasets. From left to right, input points, generated splats, and the results of the S-T cut and Normalized cut methods. How both methods retrieve similar surfaces in all the experiments can be observed.



(a) Point Set (b) S-T Cut (c) Normalized Cut

Figure 7.10: Elephant dataset (a). The results of both S-T cut (b) and Normalized cut (c) were created using only undirected normals. However, for the S-T cut case, an extention for each linear contribution had to be defined, i.e., they are splats using a jet surface of $d = 1$ approximation, as defined in Chapter 6.



(a) Point Set

(b) Normalized Cut

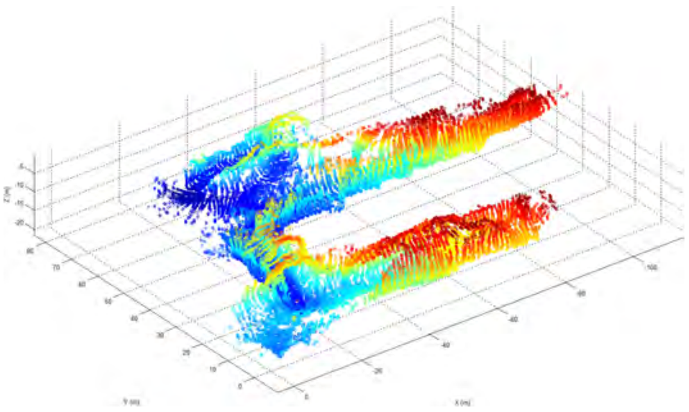
Figure 7.11: Underwater Mound retrieved with a multibeam survey (a), and the reconstructed surface using Normalized cuts (b).

the mono-beam rotating sonar head was positioned orthogonally to the cave, so that a single scan provides a 360^O view of its walls. Additionally, in this case the trajectory was optimized a posteriori through a SLAM approach [152]. Figure 7.12 shows the dataset along with the reconstruction and a cross section intersecting the reconstructed model, which allows the viewer to understand the shape of the interior of the cave. Note that small details, like the small tunnel visible in the upper part of Figures 7.12 (b,c), are faithfully recovered despite the sparse sampling.

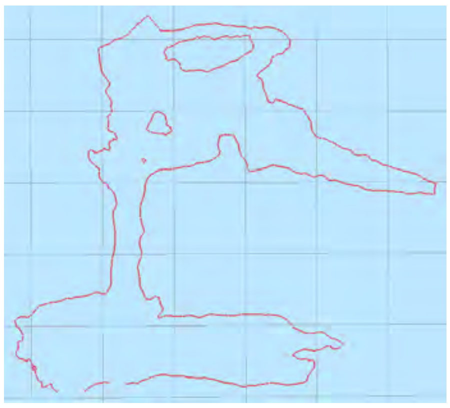
Focusing now on multi-view stereo datasets, we test the presented methods against the point sets obtained using a plane sweeping methodology. We have seen in previous chapters that these point sets present a larger quantity of aberrations, and that their processing was problematic for the methods proposed in the previous chapters. We will see how the change from a local view (splats creation and intersection) to a global view (signed distance function) introduces robustness into the surface reconstruction approach. More precisely, since the surface is defined as the interphase between two well-defined volumes, we obtain the advantage of being able to extract perfectly manifold surfaces, which was not ensured in the previous methods without post processing steps. Note, however, that here we only confront the noise reduction problem, since gross outliers are eliminated during the splat creation step. Unfortunately, as mentioned above, the S-T cut method cannot be used in some of the cases because of the noisy splat representation retrieved.

Let us start with the Column Capital dataset. Note in Figure 7.13, how by using the methods proposed in this chapter, we can recover a larger part of the surface than when using the method from Chapter 6. Another example of a pedestrian dataset is that of the Fountain. As opposed to Chapter 6, the point set was generated using a naive plane sweeping algorithm (note the large quantity of outliers in the data). In this case, the splats generation step eliminated most of the outliers in the point set, but some of them were structured following the same plane, thus generating some erroneous splats (note how these splats are larger in Figure 7.14 (b)). Despite the aberrations in the splat representation, the Normalized cut method is able to extract a manifold surface in both cases (Figures 7.13 (b) and 7.14 (c)).

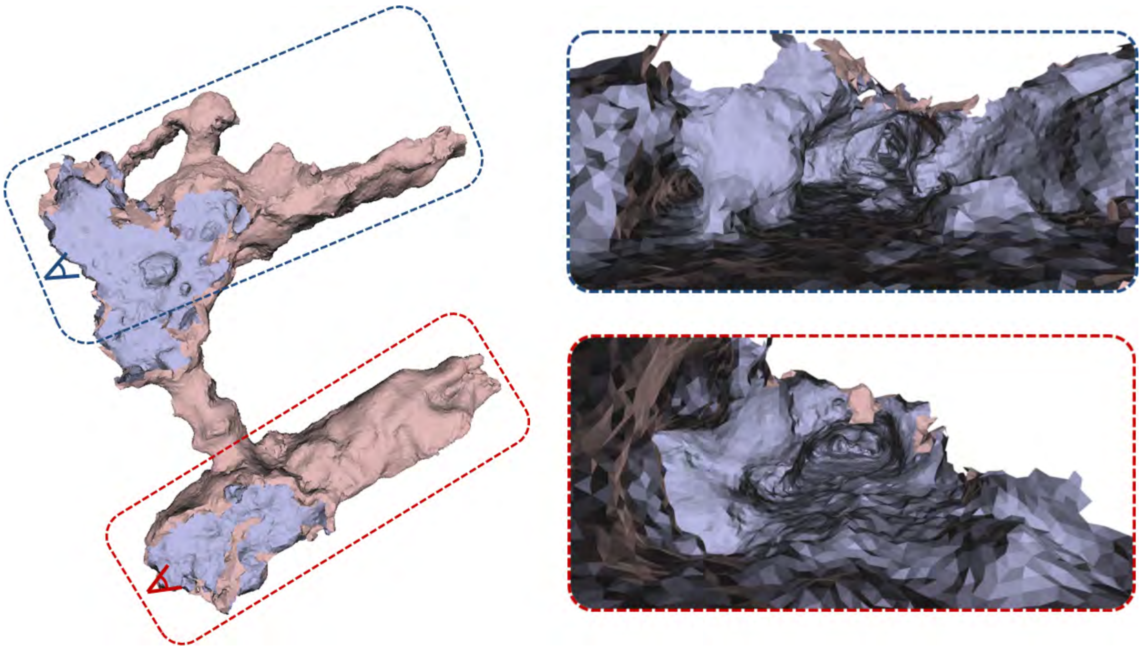
Regarding underwater datasets, we again use the ones presented in previous chapters, starting with those containing fewer outliers: the Shallow Water (Figure 7.15) and Coral Reef (Figure 7.16) datasets. In both cases, given the low number of outliers mixing up with the noise and double contours in the dataset, the splat representation obtained is suitable for both S-T cut and Normalized cut methods, and the results are similar in both cases. One may observe a slightly larger reconstructed area in the Normalized cuts case, as well as some small holes that appear in the S-T cut result. Observe in Table 7.1 that



(a) Point Set

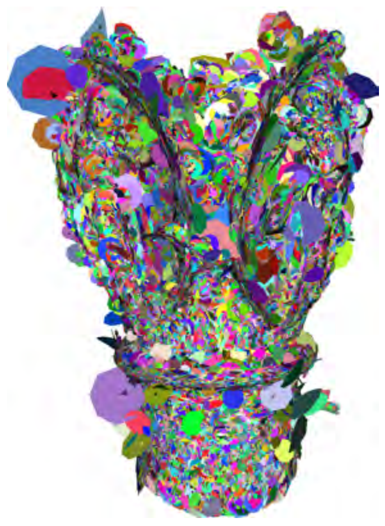


(b) Cut Section



(c) Normalized Cut

Figure 7.12: Profiling sonar survey of an underwater cave (a). In (c) one can see the reconstruction obtained by the Normalized cut algorithm, with close-ups of the two main tunnels highlighted. The view direction is also marked on the right part. Additionally, in (b) one can see a cross section of the cave to better understand the complexity of the area.


(a) Splats

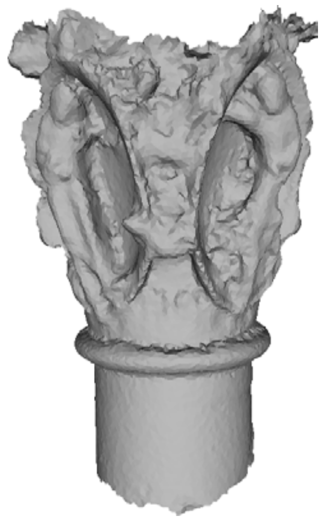
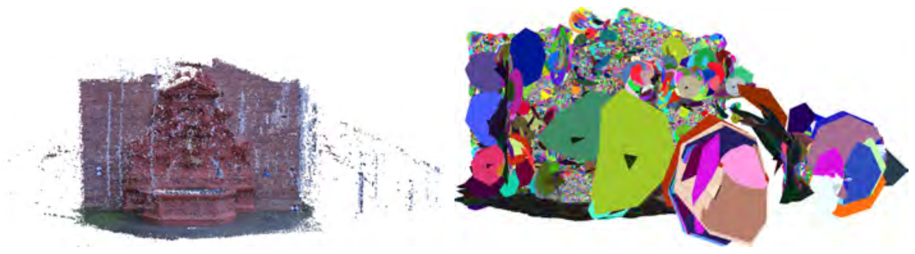

(b) Normalized Cut

Figure 7.13: Column Capital (see the original point set on Figure 6.18, page 202). Splat representation in (a) and Normalized cut surface in (b).


(a) Point Set (b) Splats

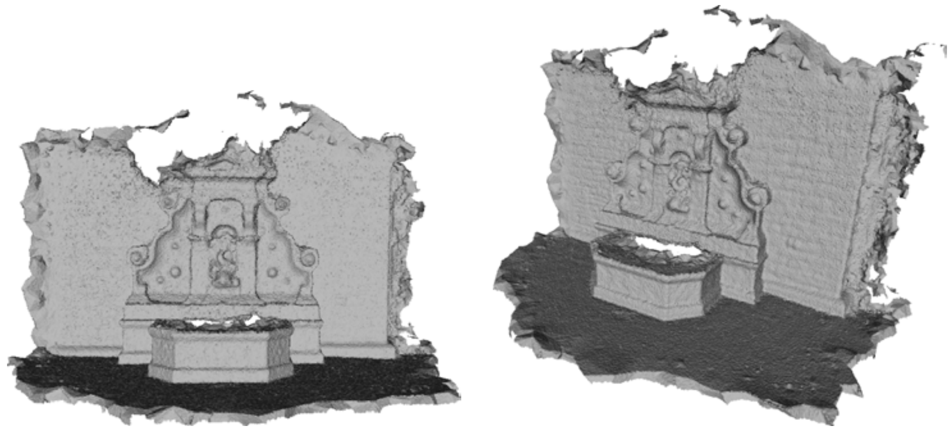
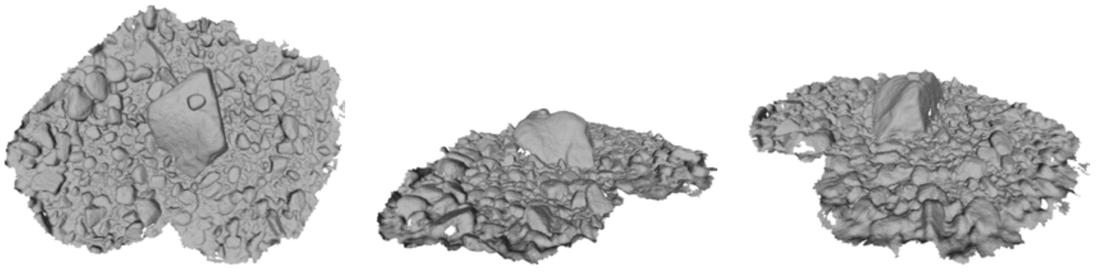
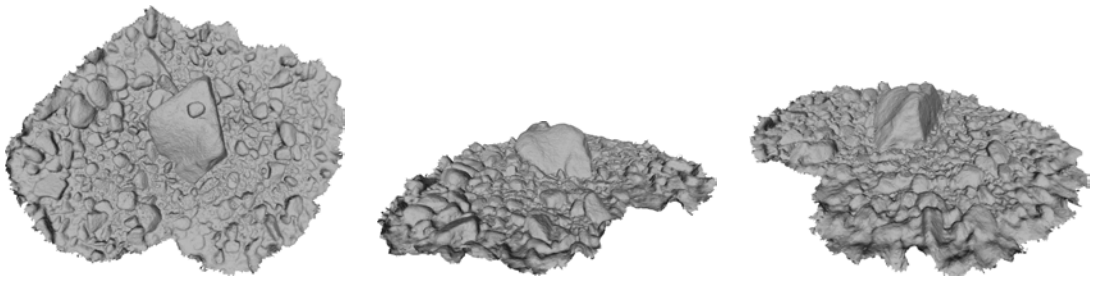

(c) Normalized Cut

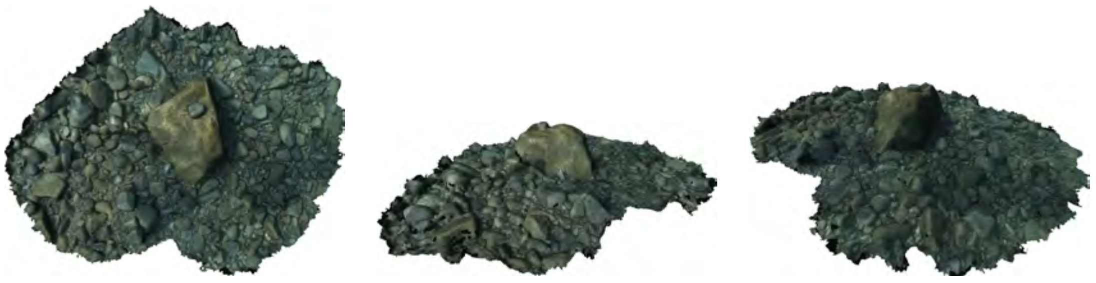
Figure 7.14: Fountain dataset, obtained using plane sweeping, obviously containing a larger number of aberrations than in the PMVS case visible in Figure 6.14 (page 198). The surface retrieved using our Normalized cut method is shown in (b).



(a) S-T Cut



(b) Normalized Cut

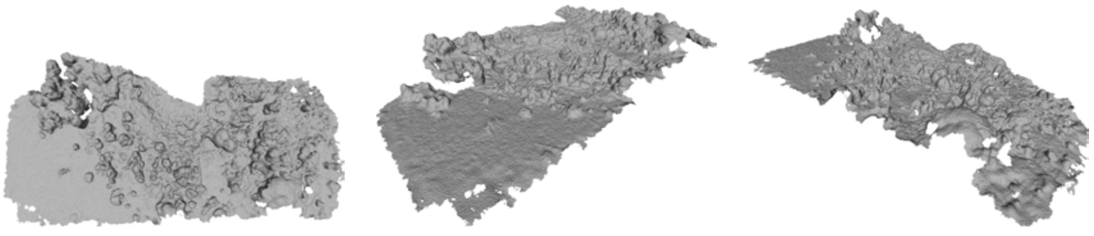


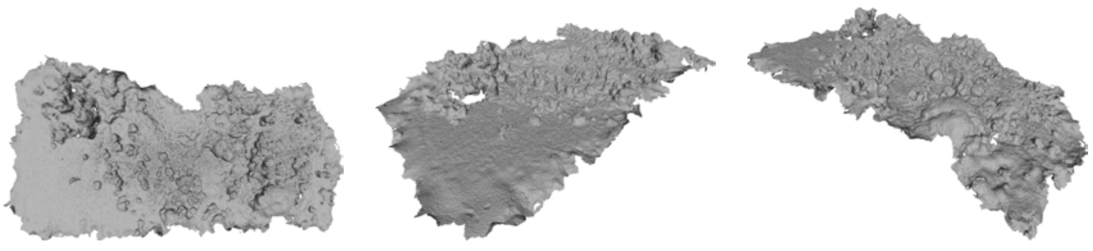
(c) Normalized Cut, with Texture Mapping

Figure 7.15: Results of the S-T cut and Normalized cut methods for the Shallow Water dataset in figures (a) and (b) respectively. For an improved visualization of the reconstructed area, we provide the texture mapped version of the Normalized cut result in (c).

these two examples are the only ones using the secondary σ_2 for the S-T cut case, since a small σ leads to an overly incomplete reconstruction with many holes in the surface. As previously mentioned, by using a larger secondary band σ_2, we obtain a larger coverage of the area.

In the case of the Tour Eiffel dataset, we are also able to retrieve a perfectly manifold surface approximating the shape of this underwater chimney. However, we can see in Figure 7.17 how a larger part of the object is retrieved in the Normalized cut case. This is due to the sparse sampling of the left-most part of the input point set, presented in Figure 5.17 (page 169), which results in a complex splat representation, visible in Figure 6.21 (a) (page 205). Consequently, this non-conforming splat representation causes the inside/outside guess to fail in some cases for the S-T cut method, dooming the global


(a) S-T Cut


(b) Normalized Cut

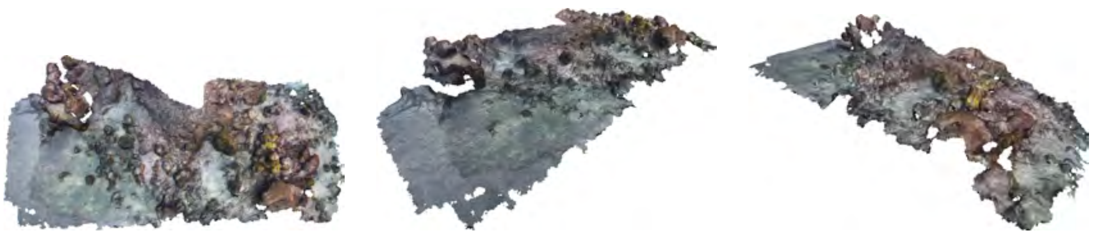
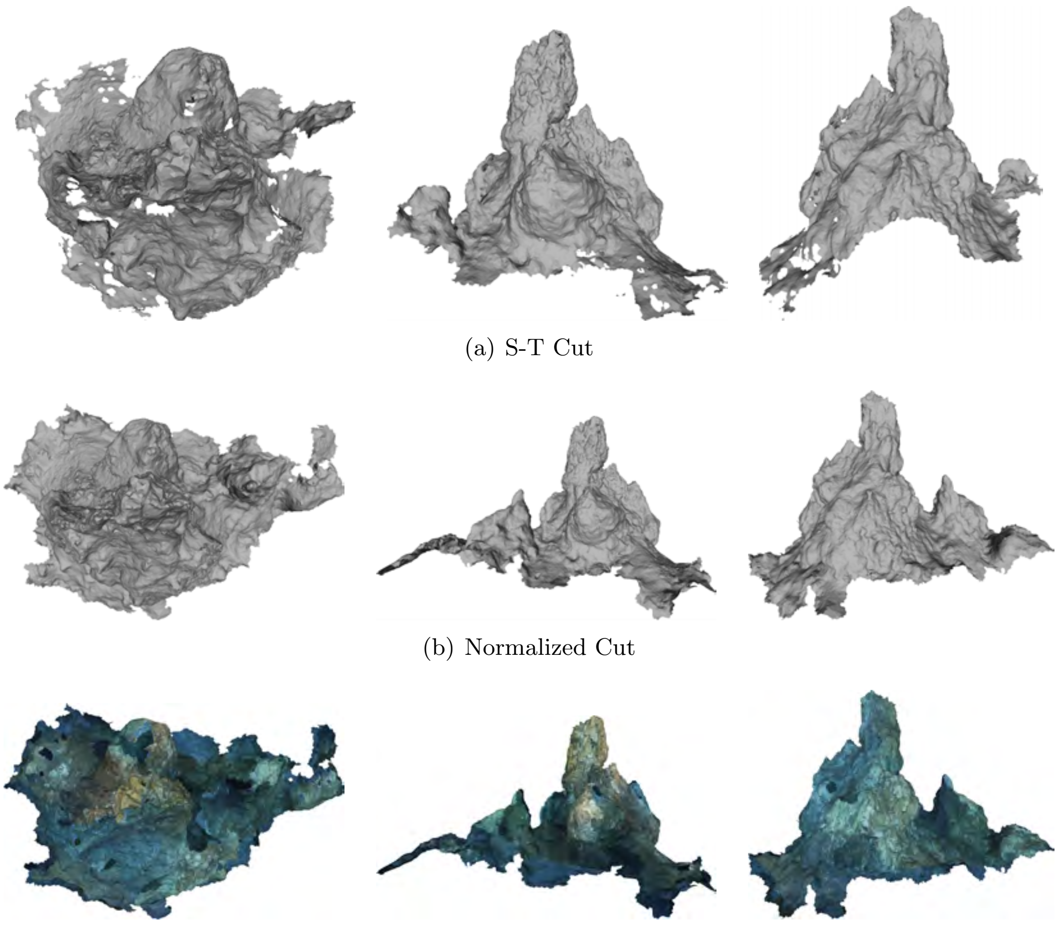

(c) S-T Cut, with Texture Mapping

Figure 7.16: Coral Reef dataset, using S-T cut (a) and Normalized cut (b) algorithms. A texture mapped version of the S-T cut result is shown in (c).



(a) S-T Cut

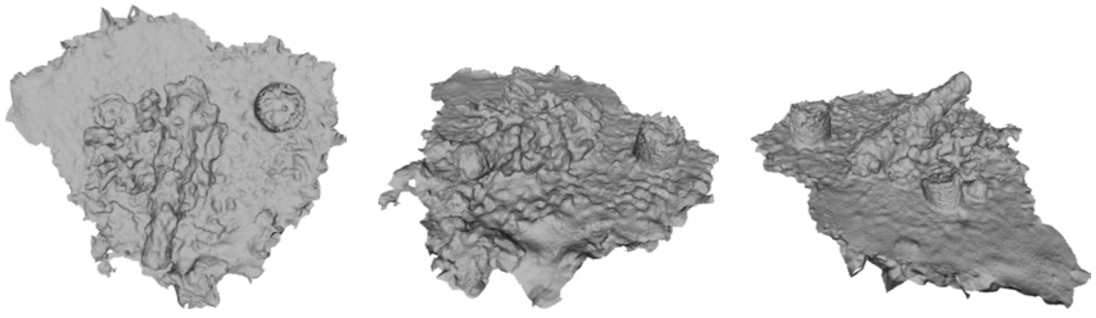
(b) Normalized Cut

(c) Normalized Cut, with Texture Mapping

Figure 7.17: Results for the Tour Eiffel dataset. From left to right: top, front and back views of the reconstructed models. We show the results of the S-T cut (a) and the Normalized cut (b) algorithm, along with its texture mapped version (c).

optimization to consider this area as part of the outside of the shape and thus not reconstructed. Nevertheless, there is a high level of detail in both cases, outperforming those obtained by methods in previous chapters.

Finally, we end this section with the dataset of the survey of La Lune shipwreck [93]. Recall the high level of corruption and the difficulties of previous methods to obtain a faithful reconstruction of this specific dataset. In addition to the one already presented in Figure 7.18, we include a survey from a different part of this same interest area, presenting another cannon and some cauldrons (Figure 7.19). In both cases, the shapes of the objects in the scene are indistinguishable either in the point set or in the splat representation (Figures 5.21 and 6.22 on pages 172 and 206, and Figure 7.19 (a,b)), but they are clearly visible in the surface reconstruction obtained with the Normalized cut (Figures 7.18 (b) and 7.19 (c)). Obviously, as previously noted, our robust intersection test fails when


(a) Normalized Cut

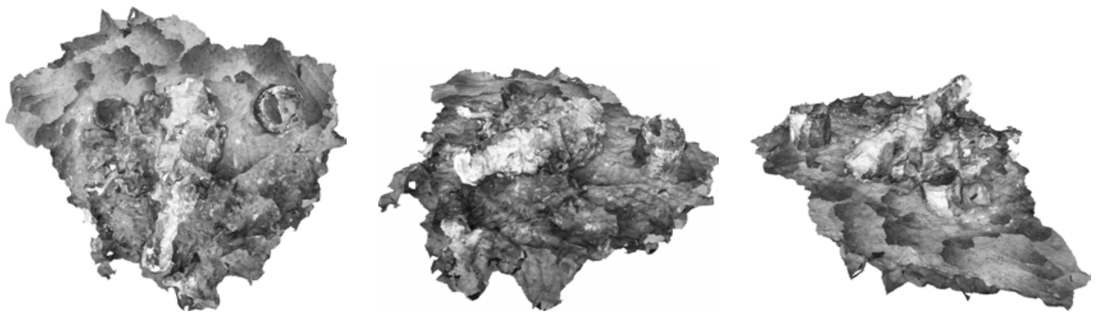

(b) Normalized Cut, with Texture Mapping

Figure 7.18: La Lune, the first dataset (presented in previous chapters). We show the reconstructed surface using Normalized cut in (a), and its texture mapped version in (b).

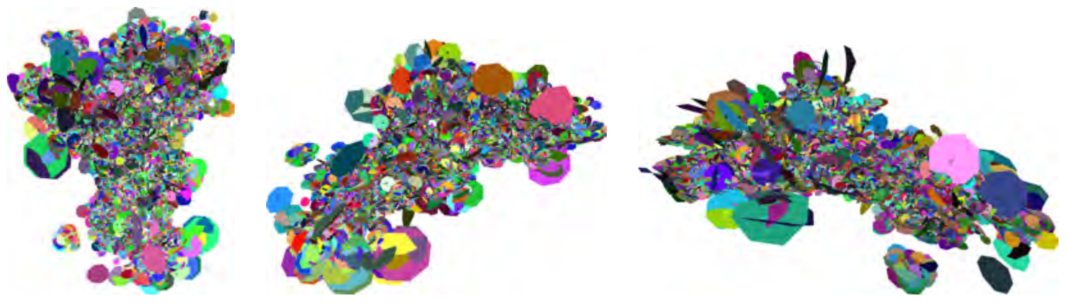
applied to highly non-conforming splats, which prevents the use of the S-T cut method in this case. Note also, in Figure 7.18, that we attain a far finer scale and, consequently, a more detailed model than using the method from Chapter 5.

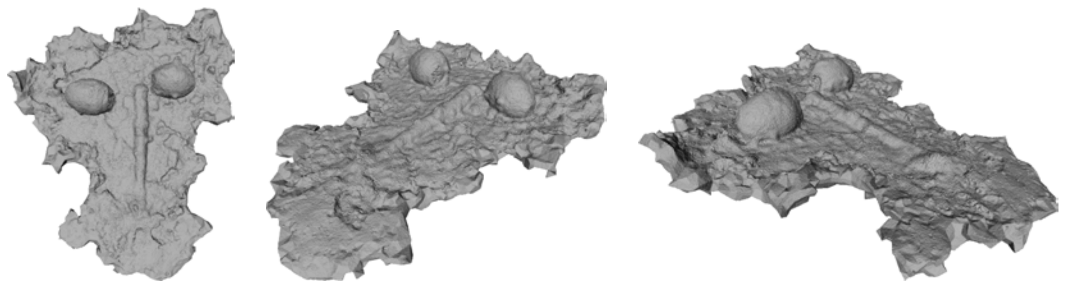
7.7 Conclusions and Future Work

We have presented two volumetric surface reconstruction methods based on minimum cuts on graphs. By making an analogy with a binary partitioning problem, we use the minimum cut along an unsigned distance function to promote its signing. This UDF is defined from the splat representation presented in the previous chapter, and is discretized in a tetrahedral grid adapted to the density of the input points. Merging the different contributions of the splats in a global view allows the mitigation of spurious splats that may remain near the surface, and extracting the surface as the zero isovalue in a well-defined volume results in the retrieved surface being manifold. Furthermore, and as opposed to most methods in the state of the art, both our proposals are designed to handle bounded surfaces.

While sharing the distance function definition, we divided the methods according to their graph representation and cutting technique. In both cases, the base graph repre-

(a) Point Set


(b) Splats


(c) Normalized Cut

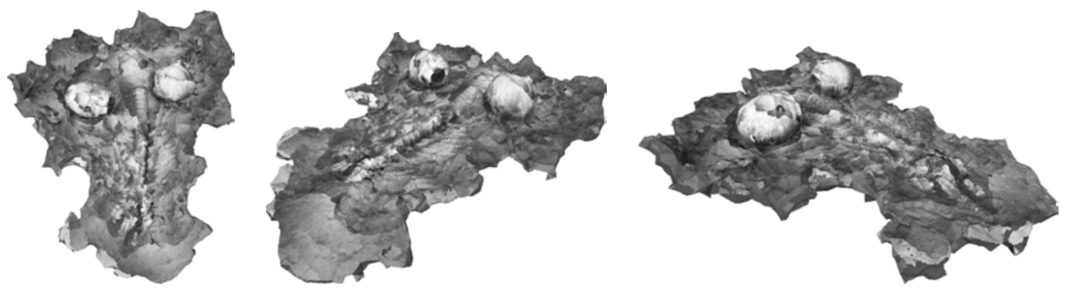

(d) Normalized Cut, with Texture Mapping

Figure 7.19: La Lune, second dataset. The point set in (a), and its splat representation in (b) show how this dataset reproduces the same level of corruption of the first dataset. The Normalized cut surface (c), and its texture mapped version (d), reveal two cauldrons located on either side of a cannon.

Table 7.1: Table showing the parameters used and the running times required to generate the results for the S-T cut algorithm in this section. Some of the parameters presented in the text have been fixed for all the datasets: $\beta = 4$, $\alpha_{re} = 1.5$, $N_{rays} = 50$, and for experiments having $\sigma_2 > 0$, the number of neighbors in the secondary band taken into account is $k_\sigma = 25$. All results were generated on a Quad-Core AMD Opteron Processor 8378 with 128 Gb of RAM.

Name	**Figure**	**Parameters**				**Run Times (s)**		
		σ *	σ_2	δ_r †	α_r/α_d ‡	UDF §	SDF ¶	Mesh ‖
Stanford Dragon	7.8(c)	6	0	0.001	0.001	6055.22	8568.24	72.37
Amphora	7.9(a)c3	3	0	0.7	0.5	1455.03	11271.80	176.55
Gargoyle	7.9(b)c3	3	0	0.7	0.5	2307.44	5051.47	196.26
Stanford Bunny	7.9(c)c3	2	0	0.001	0.001	498.88	6645.00	42.10
Stanford Armadillo	7.9(d)c3	3	0	0.7	0.5	118.72	297.23	102.01
Elephant	7.10(b)	6	0	1	1	6805.52	1953.43	142.54
Shallow Water	7.15(a)	3	15	1	0.05	3331.84	1921.37	429.87
Coral Reef	7.16(a)	3	15	0.7	0.7	2945.02	1999.66	846.76
Tour Eiffel	7.17(a)	7	0	0.05	0.05	7381.79	7315.10	170.85

*Here σ and σ_2 are expressed in terms of the average spacing between points (with $k = 6$).
†RANSAC distance threshold, as defined in Section 6.3.2.
‡Meshing parameters, as described in section 3.4.2. We omit α_a, as we fix it to $\alpha_a = 0$.
§Creating the UDF.
¶Signing the UDF.
‖Surface meshing of the SDF.

sentation is derived from the adaptive grid storing the UDF. On the one hand, we have presented the S-T cut method, which needs an initial guess for inside/outside for some of the vertices in the graph to then propagate these labels following the smooth weights governed by the UDF. On the other hand, we have introduced the Normalized cut method, which only uses the smooth weights to define a minimum cut balancing the cost of the two volumes after the partition. In both cases, we have shown with many examples that the retrieved surfaces are similar.

When comparing both proposals, we have proven that the Normalized cut method is more versatile, as it does not require any additional knowledge such as the inside/outside guess for the S-T cut case. We have noticed that when using Normalized cut, we can overcome the limitation posed by noisy self-intersecting splats that prevented the method in Chapter 6 to provide a coherent surface. However, this type of data is still not solvable using the S-T cut method, as it depends on the same RANSAC-based robust intersection detection procedure. On the contrary, parameter tuning is more sensible for Normalized cuts than for S-T cuts (as depicted in Tables 7.1 and 7.2).

A drawback of both methods when compared to the ones presented in previous chapters is their increase in memory requirements. Despite the fact that we use an adaptive grid, in both cases the storage of the distance function in a tetrahedralization represents a

Table 7.2: Table showing the parameters used and the running times required to generate the results of the Normalized cut algorithm in this section. As in the S-T cut case, we fix $\alpha_{re} = 1.5$ for all the experiments. All results were generated on a Quad-Core AMD Opteron Processor 8378 with 128 Gb of RAM.

Name	**Figure**	**Parameters**					**Run Times (s)**		
		σ [*]	σ_2	k_σ [†]	β [‡]	α_r/α_d [§]	UDF [¶]	SDF [‖]	Mesh [**]
Risu3	7.6(c)	5	10	5	4	0.1	64.50	18.45	70.87
Max Planck	7.7(b)	3	15	15	4	1	182.22	112.00	303.79
Stanford Dragon	7.8(d)	5	50	15	4.5	0.001	4884.76	9501.42	140.35
Amphora	7.9(a)c4	3	25	5	4	0.5	1674.40	4194.13	273.96
Gargoyle	7.9(b)c4	3	25	5	4	0.5	2236.00	5157.33	252.52
Stanford Bunny	7.9(c)c4	3	15	5	4	0.001	658.79	1244.07	77.43
Stanford Armadillo	7.9(d)c4	3	15	5	5	0.5	137.78	105.41	141.80
Elephant	7.10(c)	6	50	50	4	1	6579.27	7450.58	151.56
Cave	7.12(c)	3	20	5	8	0.5	96.91	177.27	31.57
Column Capital	7.13(b)	6	35	5	4	0.01	1773.45	820.85	53.07
Fountain	7.14(c)	6	50	25	6	0.025	1874.52	1268.59	435.09
Shallow Water	7.15(b)	6	50	25	8	0.05	7842.29	7611.70	305.19
Coral Reef	7.16(b)	6	100	25	8	0.05	7059.53	4831.83	1203.25
Tour Eiffel	7.17(b)	6	150	25	8	0.05	8288.46	2847.91	89.58
La Lune (1)	7.18(a)	15	50	50	9	0.05	72418.50	3762.58	533.44
La Lune (2)	7.19(c)	15	50	50	8	0.05	34296.80	2036.20	317.51

[*] Here σ and σ_2 are expressed in terms of the average spacing between points (with $k = 6$).

[†] The number of nearest neighbors taken into account to compute the UDF in the σ_2 band.

[‡] Smooth prior power.

[§] Meshing parameters, as described in section 3.4.2. We omit α_a, as we fix it to $\alpha_a = 0$.

[¶] Creating the UDF.

[‖] Signing the UDF.

[**] Surface meshing of the SDF.

large amount of memory. Furthermore, there is an inherent redundancy of splats in our representation, inherited from the redundancy requirements of the algorithm in Chapter 6. For the methods presented in this chapter, redundant splats could be simplified in order to alleviate the computational cost of computing the UDF. Thus, simplification measures for the splat representations should be proposed as future work.

Chapter 8

Quantitative Evaluation

8.1 Introduction

Through this thesis, we have encouraged the visual evaluation of surface reconstruction methods by presenting different views of the resulting mesh. While a qualitative evaluation can give a straightforward overview of the behaviour of each method, the comparison of its performance against other methods in the state of the art requires a more objective estimation. The present chapter goes in this direction by providing a numerical comparison between the results obtained by the methods presented in this thesis and those in the state of the art.

It is worth noting that quantitative evaluations are not usually discussed in the surface reconstruction literature. This is due to the lack of a consistent ground truth against which the authors could evaluate the results of their methods. This situation may have changed recently due to the creation of the reconstruction benchmark by Berger et al.[22], which provides a set of tools to both generate ground truth data and evaluate the results obtained by each method. Given its recent release, and to the best of our knowledge, the only method using it in its evaluation is the Screened Poisson by Kahzdan and Hoppe [119].

Several tests are performed with a representative set of methods from the state of the art. We start by using real data gathered using the principal scanning technologies tested in previous chapters, namely range scanning and dense photogrammetry. These datasets are used to evaluate the methods' behaviour when applied to different scanning methodologies. Then we use synthetic data generated with the benchmark by Berger et al. [22]. The original evaluation is enriched by adding not only the methods presented in this thesis to the comparison, but also other methods in the state of the art. We begin by evaluating the methods against the synthetic datasets provided by the authors and used in the original evaluation. Afterwards, we test the methods' resilience to noise by generating point sets with varying noise levels. Finally, we check the performance of the methods against outliers.

8.2 Tested Algorithms

To start with, we enumerate the algorithms tested in this chapter. In the original benchmark, only approximation-based algorithms requiring oriented point sets were evaluated. The methods presented in this thesis work on raw point sets, so it is important to compare them against a broader spectrum of inputs. Fortunately, the benchmark of Berger et al. [22] is defined in a way that does not forbid its evaluation with algorithms working with raw (non-oriented) point sets. In the following we present the 19 methods evaluated (in brackets, when different from the original, we write the acronym appearing in the figures and referred to in the text):

1. **Point Set Mesher (PSM):** The algorithm presented in Chapter 5.
2. **Splats Mesher (SM):** The algorithm presented in Chapter 6.
3. **Splats Distance S-T (SD S-T):** The S-T cut variant of the graph-based algorithm presented in Chapter 7.
4. **Splats Distance Normalized Cut (SD N):** The Normalized cut variant of the graph-based algorithm presented in Chapter 7.
5. **Poisson [118]**.
6. **Screened Poisson [119] (SPoisson)**.
7. **Fourier Transform [117] (FFT)**.
8. **Wavelets [153]**.
9. **Smooth Signed Distance [41] (SSD)**.
10. **Point Set Surfaces [6, 2] (PSS)**.
11. **Implicit MLS [124] (IMLS)**.
12. **Algebraic PSS [97] (APSS)**.
13. **Markov Random Field [169] (MRF)**.
14. **Multilevel Partition of Unity [165] (MPU)**.
15. **Smooth PU [161] (SPU)**.

16. **Multilevel Compact RBF [166] (Multi-RBF).**
17. **Spherical Cover [168] (SC).**
18. **Robust Cocone [65] (RC).**
19. **Power Crust [13] (PC).**

Note that the ten algorithms tested in the original evaluation [22] are also presented here. However, we enrich the original evaluation by adding three approximation-based techniques (SPoisson, SSD and MRF), and two more interpolation-based methods (RC and PC). Even though the difference in nomenclature between the algorithms might be misleading when compared to that of the original reference [22], we decided to give the algorithms a name more coherent with their presentation in Chapter 4.

8.3 Real Data

We start by evaluating the behaviour of the algorithms when applied to real datasets. For this purpose, we reproduce the experiments proposed by Kazhdan and Hoppe [119], consisting of randomly dividing the input points into two equally-sized sets. Given this partition, one of the sets is used for evaluation while the second is used for validation. In this way, we use the evaluation set as input to the algorithms, and the validation set to compute the distances to this reconstructed surface.

Note that, since we are using real data, the input point sets contain some noise, which depends on the scanning methodology used to retrieve the point set. However, we filter outliers out of these datasets, since including them would corrupt the distance computation. Obviously, the tests measure how good the reconstruction is when compared to the original data. Thus, we evaluate the overall behaviour of the method against the input data, but we cannot draw global conclusions as no ground truth is available.

Regarding the tested datasets, we use two range scans –Bunny and Amphora–, acquired in a controlled environment, and two point sets retrieved using dense photogrammetry –Tour Eiffel and Shallow Water–, obtained using different methodologies. On the one hand, the Bunny and Amphora datasets denote the behaviour of the methods under good scanning conditions. Despite the datasets containing some noise, they present a dense sampling, easing the reconstruction of the surface. However, they differ in noise scale and complexity. The Tour Eiffel and Shallow Water datasets provide a notion of the application of the methods on the type of input expected in this thesis, i.e., point sets obtained through dense multi-view stereo methods. However, in the Tour Eiffel dataset case, the point cloud was generated through a greedy dense reconstruction process [84], which results in a less noisy point set, but with large undersampled parts (i.e., holes in the surface). On the other hand, the Shallow Water dataset was obtained through a plane sweeping algorithm [209], providing a denser but also noisier point set. Note that, since the plane sweeping method does not provide normals on the point sets, we compute them for the methods required using Hoppe's algorithm [104], with a neighborhood of $k = 100$.

8.3.1 Results

We present the results with respect to the mean distance obtained from the validation set of points to the reconstructed surface, computed using the tools available in MeshLab [53]. As noticed in Kazhdan and Hoppe [119], the fact that the validation points are noisy makes the commonly used Hausdorff distance unreliable. Given the variable scale of the data, all the results are depicted with respect to the BBD of the model. Along with the mean distance measures, we also show a view of the resulting surface.

The Stanford Bunny is an ubiquitous model in surface reconstruction articles, so it has been most frequently used to assess a qualitative comparison between algorithm results. This is coherent with the results obtained, where we can observe how all the methods behave similarly. The mean distance obtained is comparable for most of the methods, with the exception of the larger values with the MRF and the RC method. On the one hand, the MRF results are caused by the oversmoothed results obtained. On the other hand, the RC recovers a correct global shape but did not preserve fine details, as highly curved regions are closed.

For the Amphora dataset, the model is a bit more complex and presents more noise than the Bunny dataset. Consequently, the results obtained are more variate. On the one hand, the best mean distance is obtained by the SM, SPoisson, MRF and Multi-RBF. However, these best scores are closely followed by those of many other methods with no significant differences. On the other hand, the ones obtaining worse results are FFT, PSS and SPU. While the FFT results are mainly caused by the closing of the handles on the bottle, those of PSS and SPU are caused by oversmoothing.

The Tour Eiffel dataset has the added complexity of having large undersampled areas. Note in Figure 8.3 that, since most of the methods force the shape to be closed, the resulting surfaces present some approximated parts in places with no data. Depending on the method, these surface parts may be far from realistic. However, recall that the validation points are obtained from the original point set, and thus the non-sampled areas are not taken into account in the mean distance computation. Furthermore, the sparse sampling has caused some of the parts to be unreconstructed. This is reflected in the mean distance measures, where we can see that the SD S-T, SD N, FFT and Wavelets attain a high value because of missing parts of the shape. On the other hand, oversmoothing is also penalized in the Poisson case. Nevertheless, a large number of methods, including SPoisson, SSD, PSS, IMLS, APSS and PC, attain good results in this case.

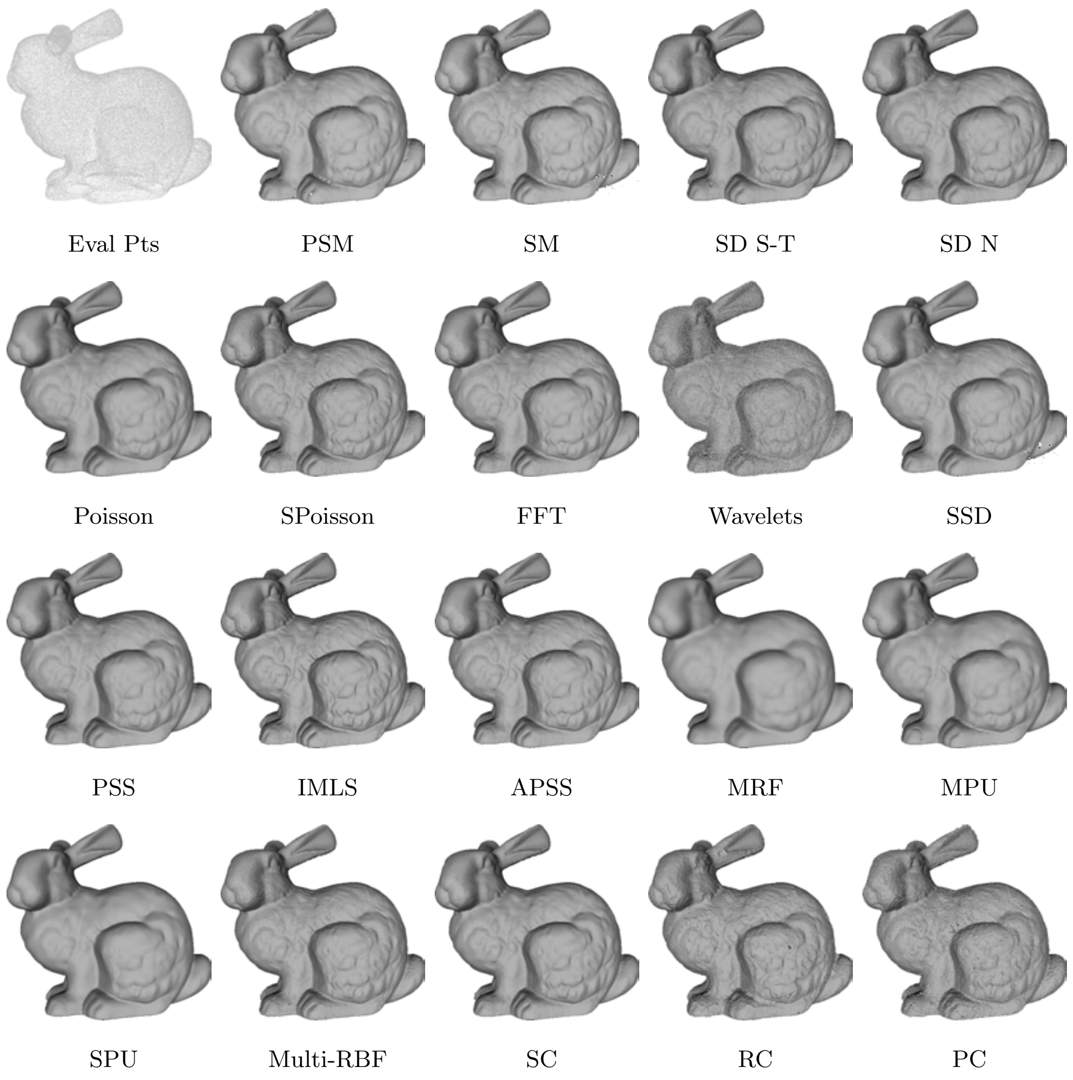
Finally, the Shallow Water dataset describes a bounded surface, and consequently some results on Figure 8.4 also present made-up parts as a consequence of the above mentioned reason. Nevertheless, all mean distances are very similar. This is caused by the good sampling provided by the point set. As in previous cases, the ones obtaining the worst measures are FFT, Wavelets and RC. On the contrary, the best results are obtained by SPoisson and PC.

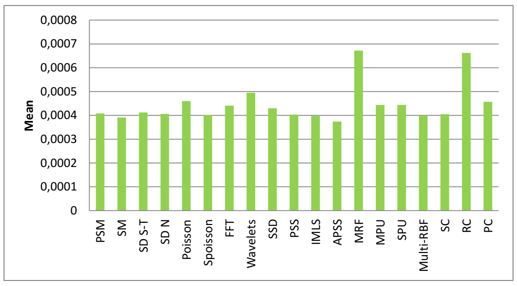
To sum up, it is obvious that the best results are obtained by SPoisson, and that the ones obtaining worse results are FFT, Wavelets and RC. The results of the rest of the methods vary depending on the dataset, but in all cases the ranges are within the same scale. This experiment shows that the results obtained for each method greatly depends on the properties of the input point set. Nevertheless, and even if variable, it is clear that the results obtained in these tests by our methods are within the ranges of those in the state of the art.

8.4 Surface Reconstruction Benchmark

The tools provided by Berger et al. benchmark [22] simulate a real triangulation-based laser range scan system. More precisely, it generates a set of samples from an implicit reference surface, simulating the impact of a virtual laser on this surface and its further projection onto the image plane of a camera. Then, given the known laser source position and its projection onto the image, the position can be triangulated. This reference implicit surface, which is a mixture of local surfaces (a variant of the MPU procedure extracted from a triangulated surfaces), is the ground truth of our object. Through ray-tracing, this MPU implicit surface is scanned from uniformly sampled positions over an enlarged bounding sphere of the object with the camera oriented towards its center of mass. A wide variety of scanning parameters can be changed during the creation of the test point sets, including varying resolution, number of range scans, the laser's field of view, the magnitude of the measurement/registration error or missing data. Unfortunately, outlier data cannot be generated with the tools provided.

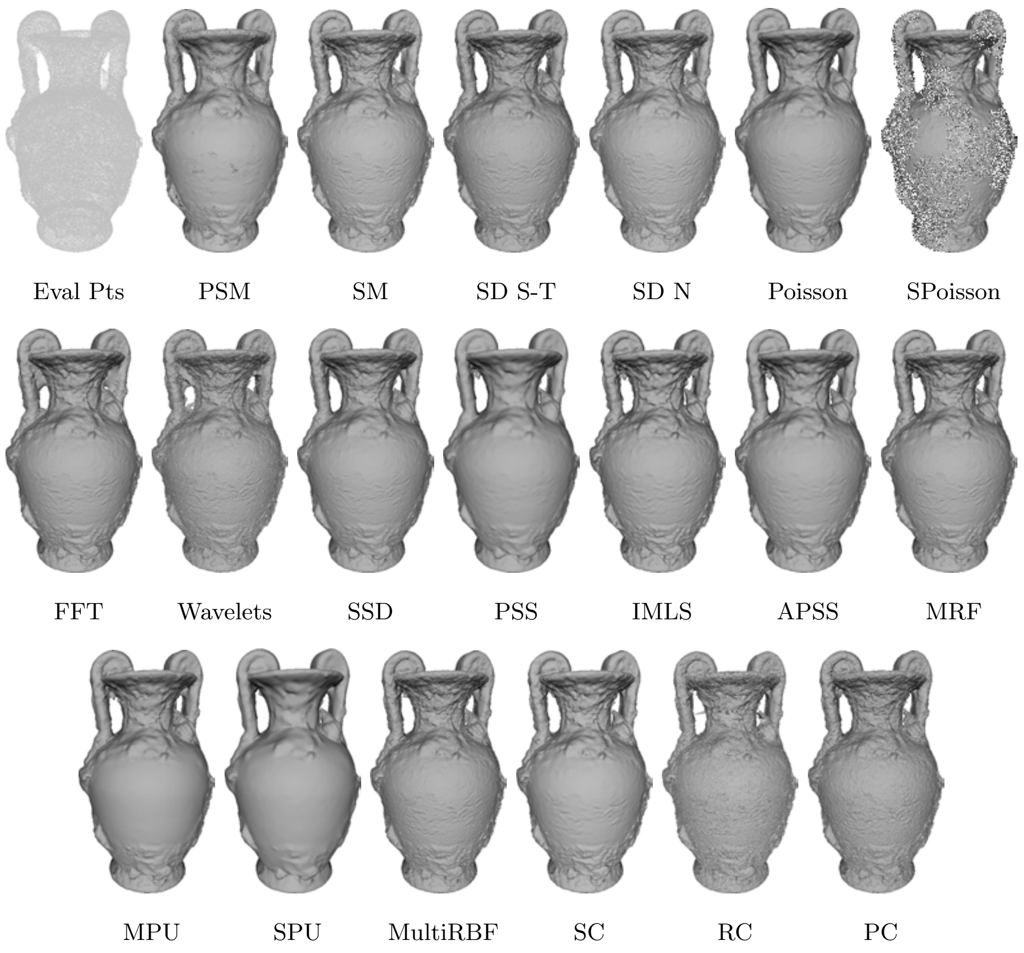
Once the test samplings are created, the different reconstruction algorithms can be applied. Then, the evaluation is computed using a bidirectional distance map. First, both the MPU implicit reference shape and the reconstructed surface are densely sampled with a dynamic particle system ensuring a close-to-uniform sampling rate. Then, a bidirectional distance map is created, mapping a set of sample points on the reconstructed surface to its closest point on the sampled original implicit surface (i.e., the ground truth), and

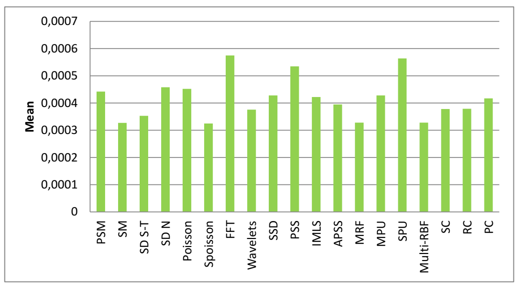




Mean Distance

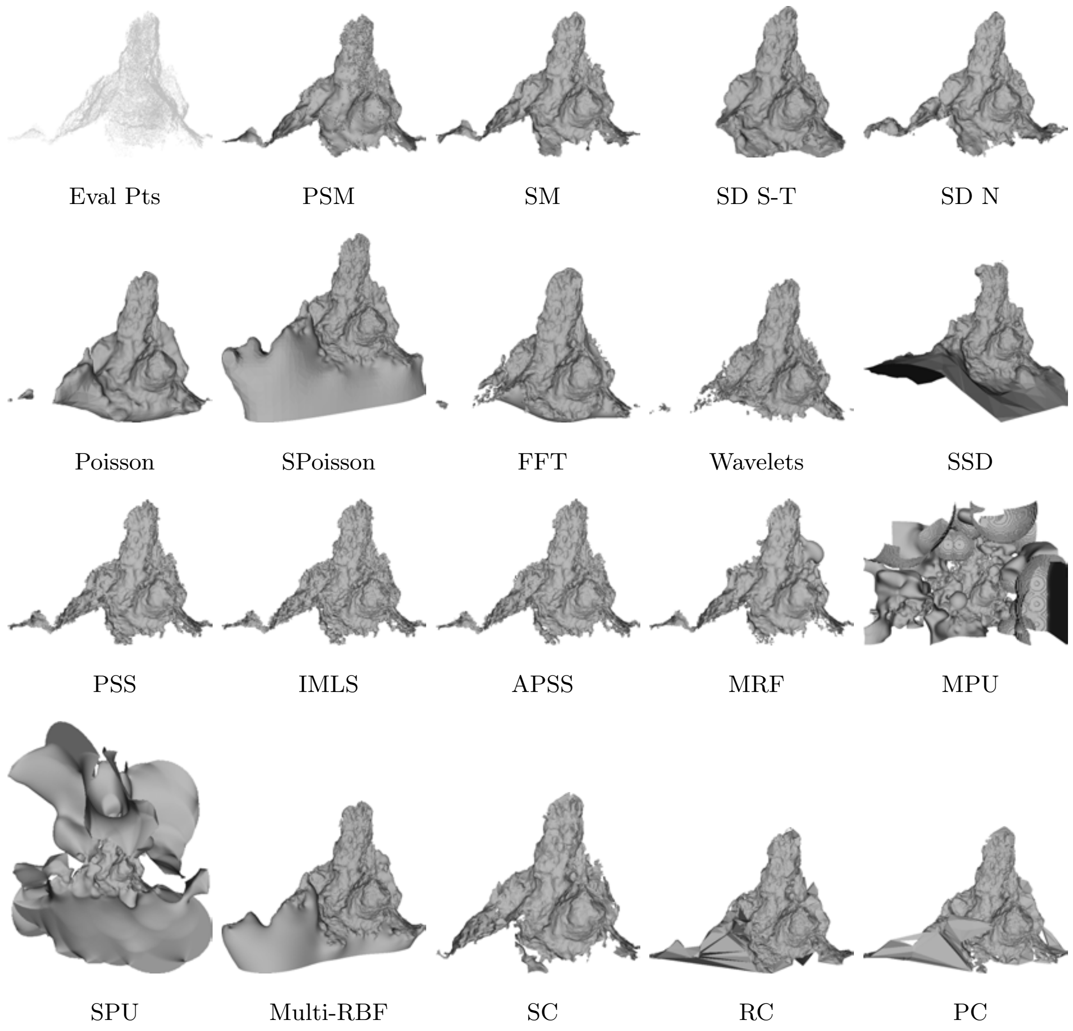
Figure 8.1: Real data tests, results for the Bunny dataset. Values with respect to the BBD.

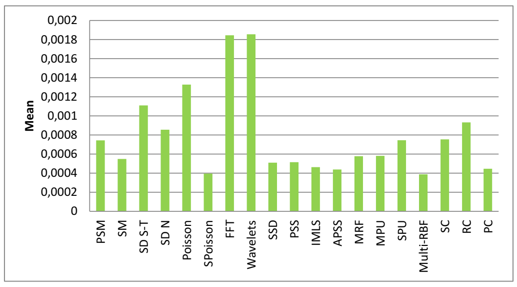




Mean Distance

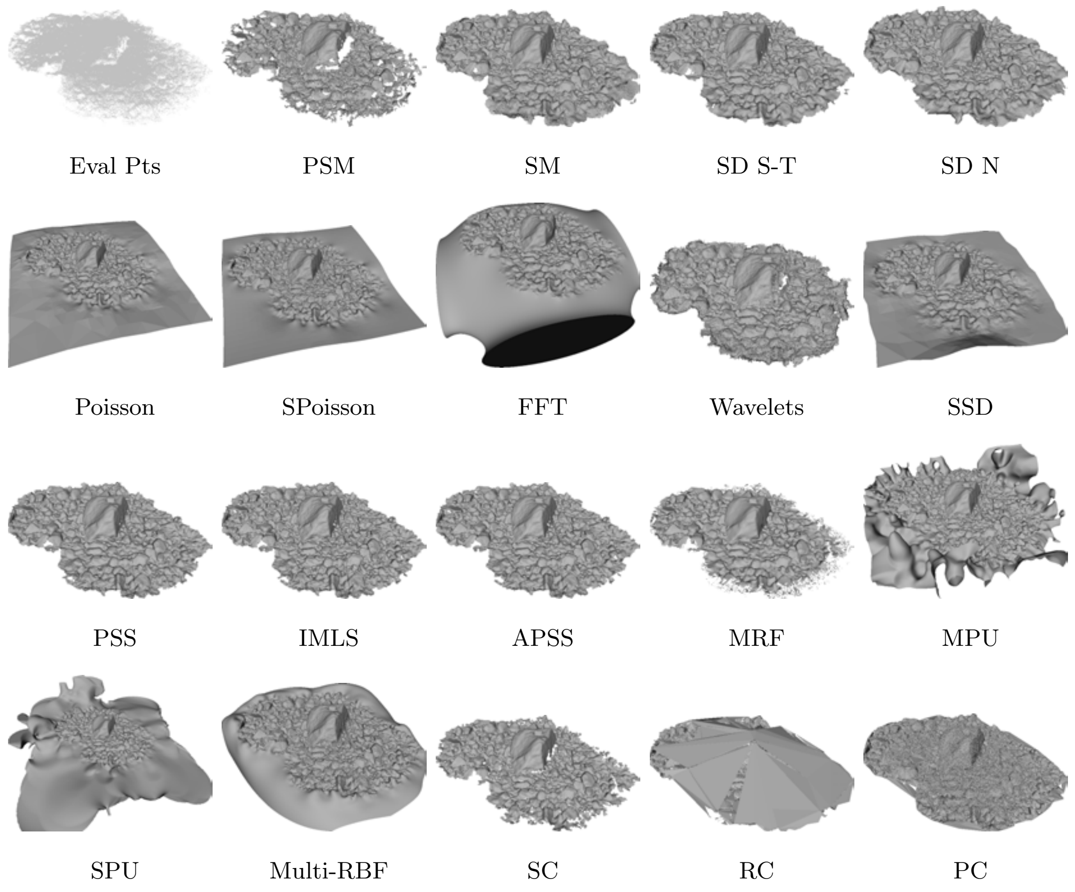
Figure 8.2: Real data tests, results for the Amphora dataset. Values with respect to the BBD.

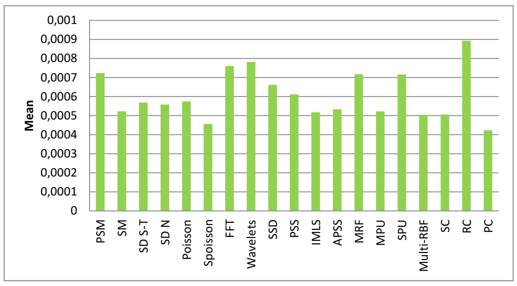




Mean Distance

Figure 8.3: Real data tests, results for the Tour Eiffel dataset. Values with respect to the BBD.





Mean Distance

Figure 8.4: Real data tests, results for the Shallow Water dataset. Values with respect to the BBD.

viceversa, and a set of error measures are computed. These error measures are **Mean distance**, **Hausdorff distance** and **Mean angle deviation**, all of which are computed bidirectionally (i.e, from the reconstructed surface to the ground truth and viceversa).

However, note that only the largest connected component is taken into account for distance evaluation, since some of the algorithms may tend to return more than a single component (usually due to noise). Furthermore, anomalies on the surface, such as non-manifold components, are included as separate parts, and thus are not contained in the largest connected component used in the validation. Obviously, this penalizes the methods promoting these configurations.

From the four experiments proposed in the original paper, we reproduce two of them: the *error distributions* and the *noise tests*. Additionally, we provide an evaluation accounting for the resilience to outliers.

Regarding the other two experiments presented in the original article [22], the *sparse sampling* and *missing data* experiments were not taken into account in our evaluation. The reason behind this decision is our focus on retrieving a bounded surface. In both experiments, they force a lack of data in the models, in the first case as a lack of sampling, and in the other as deliberate holes in the sampled surface. The objective of these experiments is to evaluate the approximative behaviour of the algorithms in parts with low or a total lack of data. From our point of view, the lack of data also indicates a lack of surface, i.e., we do not want the surface to be approximated on these parts. Consequently, these evaluations make little sense for our purposes.

8.4.1 Parameter Tuning

All the algorithms were tuned in order to provide a fair comparison. The parameters were carefully selected to provide a similar resolution for all the reconstructed shapes. For all the algorithms, we have respected the parameters recommended by the authors, when available, and tuned them when necessary to achieve a better reconstruction quality, based on visual examination of the results. In the following, we describe some specific decisions taken for some of the algorithms reviewed.

A key parameter for all the methods is the resolution of the output surface. For the methods working with implicit surfaces evaluated on a regular grid, we fixed it to a size of 350 (this includes PSS, IMLS, APSS, MRF, MPU, SPU and Multi-RBF), with the exception of the FFT method, for which we used a grid of 512. For those methods requiring an octree (i.e., Poisson, SPoisson, Wavelets and SSD), we fixed the depth to 10. Then, for our methods (PSM, SM, SD S-T and SD N), working with an RDT surface mesher, the resolution was tuned to qualitatively mimic that obtained by the other methods. For the rest of the algorithms, which are basically interpolation-based (SC, RC and PC), the resolution cannot be tuned.

Regarding algorithm-specific parameterizations, while we described a cleaning step containing a hole filling process in Section 5.4, we did not include it in the PSM case, leaving all the holes in the data. Additionally, as mentioned above, the SD S-T and SD N methods (presented in this chapter) depend on the splat representation, generated as part of the SM method. In the smooth surfaces case, the splats used for both SM and SD S-T/N are the same. On the contrary, the splats are different for the noise and outliers tests.

The tuning of other parameters available in the algorithms, which are usually related to noise attenuation, have been treated differently depending on the test performed. For the smooth surfaces test, all the reconstructions have been obtained in a batch process, and consequently, the set of parameters is fixed for a given algorithm and shape. Contrarily, for the noise and outliers test, we fine-tuned all the algorithms to produce the best results for each point set tested.

8.4.2 Smooth Surfaces

We evaluate the algorithms against the data generated for the smooth surfaces error distribution test of the original paper by Berger et al. [22]. For each algorithm applied to a single shape, 48 point sets are generated by modifying various parameters, thus simulating the variability of sampling that may appear when scanning the same object using different methodologies and conditions. Since we use the original models, already provided in the benchmark, we refer the reader to the original article [22] for a more detailed description of the parameters used to generate them.

Even though shapes/point clouds and graphical depictions of the results are provided by the authors, the numerical results are not available. Thus, we recomputed the results for all the tested algorithms. The possible variance in parameters used to reconstruct the surfaces may result in slightly different results when compared to the original evaluation. Nevertheless, overall, the results and conclusions obtained for the methods already tested in the original reference are very similar (if not the same).

Obviously, using multiple shapes with varying scanning requirements gives a global view of the behaviour of an algorithm, as it prevents obtaining biased evaluations, which may happen when a method is applied to a single point set having some specific properties favoring its performance.

More precisely, we use the three smooth sampled surfaces provided in the original paper, referred to as *Gargoyle*, *Dancing Children* and *Quasimoto*. The authors also provide two other shapes, named *Anchor* and *Daratech*, that were not taken into account. The reason is that they represent piecewise-smooth surfaces (i.e., sharp features are present in the models), which are clearly beyond the scope of this thesis.

8.4.2.1 Results

We divide the results according to each dataset using a box plot representation to describe the effectiveness of each method. As suggested in the original article, the median in those box plots is a good estimator for the overall performance of the algorithm, while the quartiles indicate the variability in the results. Please observe that, in cases where the box plot is too large and would consequently occlude the scale of the rest of the data, we cut this box. In these cases, we explain the reason behind this poor behaviour in the text.

From a global point of view, after reviewing Figures 8.5, 8.6 and 8.7, we can conclude that, in general, the methods that achieve a better (similar) overall score in the distance tests (mean/maximum distance) are SM, SPoisson, SSD, SPU and Multi-RBF. The last three also attain a good score for the mean angle deviation, but this is not the case for the SM method, where the score is higher (and consequently, worse). In fact, all the approaches proposed in this thesis (PSM, SM, SD S-T and SD N) score very badly for the mean angle deviation. However, recall that, as opposed to algorithms with a better score on this measure, our algorithms do not use oriented point sets. Consequently, none of our algorithms tries to enforce normal coherence between the oriented point sets and the recovered surface, which is what produces this decrease in performance. The same happens for the RC and PC methods, which also work with raw data and neither provides consistency of normals.

From the methods working with raw point sets not proposed in this thesis, the one scoring the best in general (but specially in the angle deviation measure) is SC, while the one obtaining the worst results is RC. We can also observe that the methods using MLS score similarly in all cases, with the APSS method performing slightly better. The PSM, SD S-T, MRF and MPU behaviours vary depending on the dataset, but still obtain good overall results. The Wavelets method tends to perform well in the Hausdorff distance case, but obtains worse results in the mean distance case, meaning that surfaces are less smooth. On the contrary, Poisson and FFT provide smooth surfaces, indicated by the higher values in the mean distance, but relatively low values in the Hausdorff distance. Finally, we can see how the method obtaining the worst overall results is the SD N. As mentioned above, the SD N method is more sensitive to parameter tuning than the rest of our proposals. Since we fixed a generic set of parameters for all the test point sets, some of the shapes have not been correctly recovered, resulting in some missing parts of the shape. Nevertheless, if we could tune the parameters for the specific shapes failing, the results would improve. Obviously, we did not try to fine tune each of the shapes separately in order to provide a fair comparison between algorithms.

While some methods score similarly for all the tested shapes, some of them obtain different marks depending on the kind of dataset they are applied to. The obvious case is the SD N case, where results are comparable to the other methods for the Gargoyle point sets, which is not the case for any other shape. Note that the scores are not that good in this case, since even if it achieves coherent results in most cases, the reconstructed surfaces may be incomplete. The SD S-T method has a larger variance in results for the Dancing Children shape, because of a problem similar to the SD N case, that is, in some situations the global parametrization failed and parts of the shape were missing in the reconstructed surface.

Disregarding the special case of the SD N method, we can conclude that the performance of our proposals PSM, SM and SD S-T is comparable to the methods in the state of the art. On the one hand, it is important to notice how our methods, working on raw point sets, obtain distance scores comparable to those methods in the state of the art using oriented points (methods with indexes 5 to 16). This evidences that normals are not necessary to obtain faithful surfaces. On the other hand, if we compare them to other methods not using normals (SC, RC and PC), we can see how our methods outperform both RC and PC. This is because there is no noise correction in these cases, and consequently, the results with noisy point sets are doomed to be worse. For this same reason, our results compare favorably with the SC case, as this method is concerned with noise attenuation.

Similarly to the original reference, we provide additional information regarding the connected components, length of boundaries and manifoldness in Table 8.1.

8.4.3 Noise Test

All the algorithms tested in this chapter claim resilience to noise to some extent. However, a clear quantization of the noise levels admitted for each method is never provided by the authors. For this purpose, we use the same noise test proposed in Berger et al. [22]. However, instead of using a simple shape like the one in the original article, we synthetically scan the more realistic shape of the Max Planck Bust (obtained from the aim@shape repository).

Starting from the original triangle mesh, we generate the reference MPU surface. Then, we synthetically scan this shape using the simulated laser-based triangulation scanner, using a consistent number of views (20) and scanner resolution (200×200) at each scan to adequately sample the shape, and enforcing no registration error or missing data. However, we do manipulate the two parameters affecting the noise on the resulting shape: the noise magnitude and the laser's field of view. On the one hand, the noise magnitude relates to a corruption in the laser projection. On the other hand, the laser's field-of-view refers to

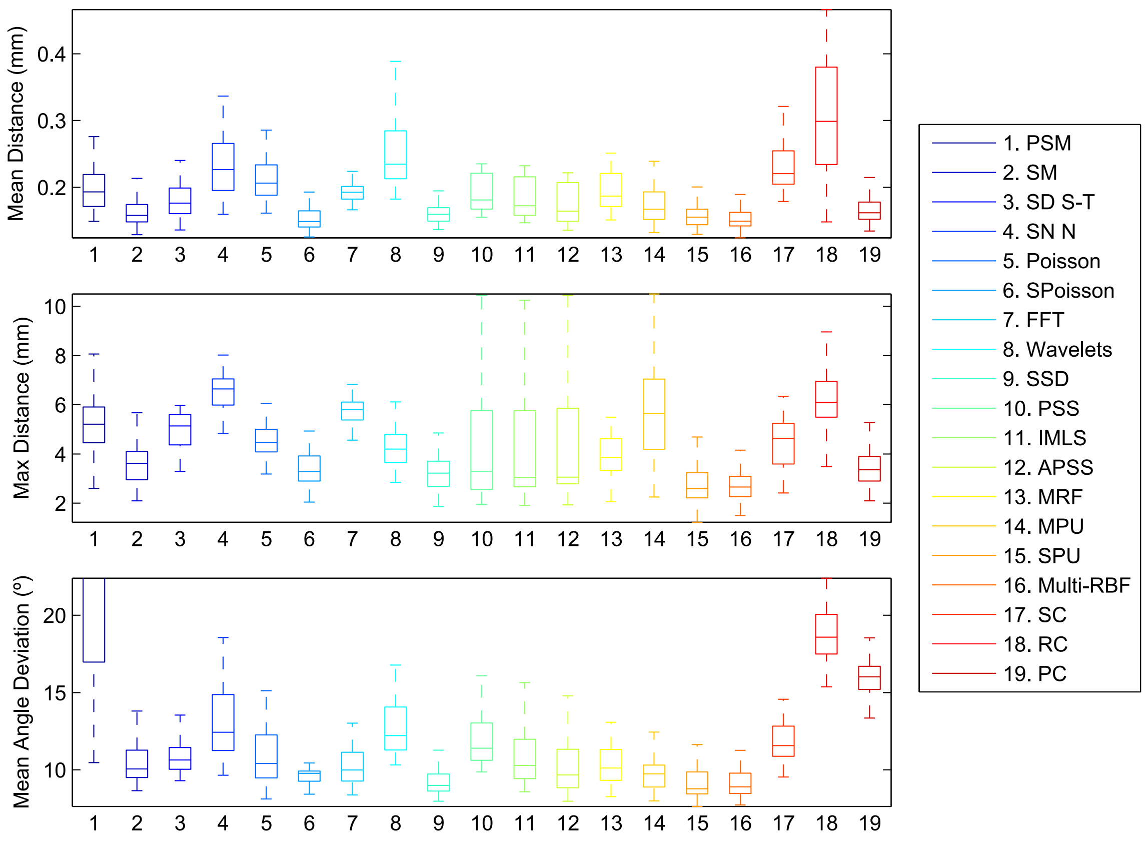


Figure 8.5: Benchmark by Berger et al.[22], results for the **Gargoyle** dataset. We show the mean and maximum (i.e., Hausdorff) distances, and the mean angle deviation for each method when applied to 48 synthetic scans generated using various scanning parameters.

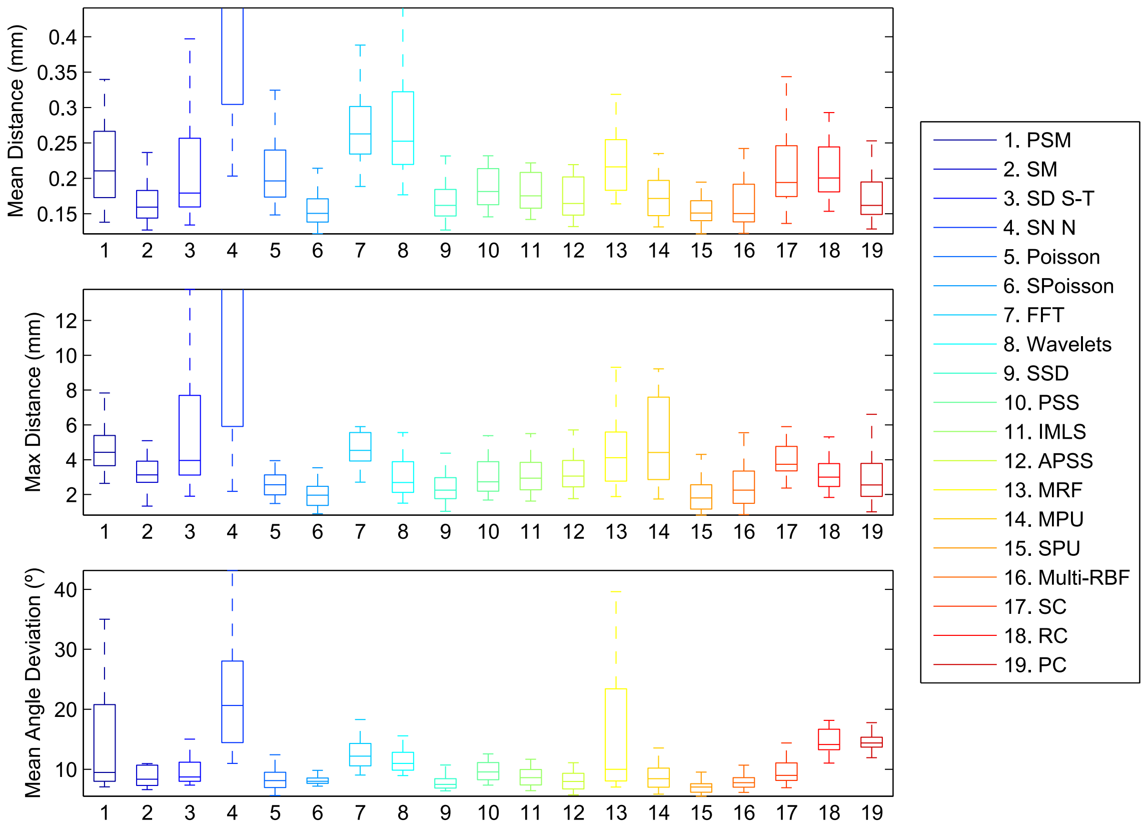


Figure 8.6: Benchmark by Berger et al.[22], results for the **Dancing Children** dataset. We show the mean and maximum (i.e., Hausdorff) distances, and the mean angle deviation for each method when applied to 48 synthetic scans generated using various scanning parameters.

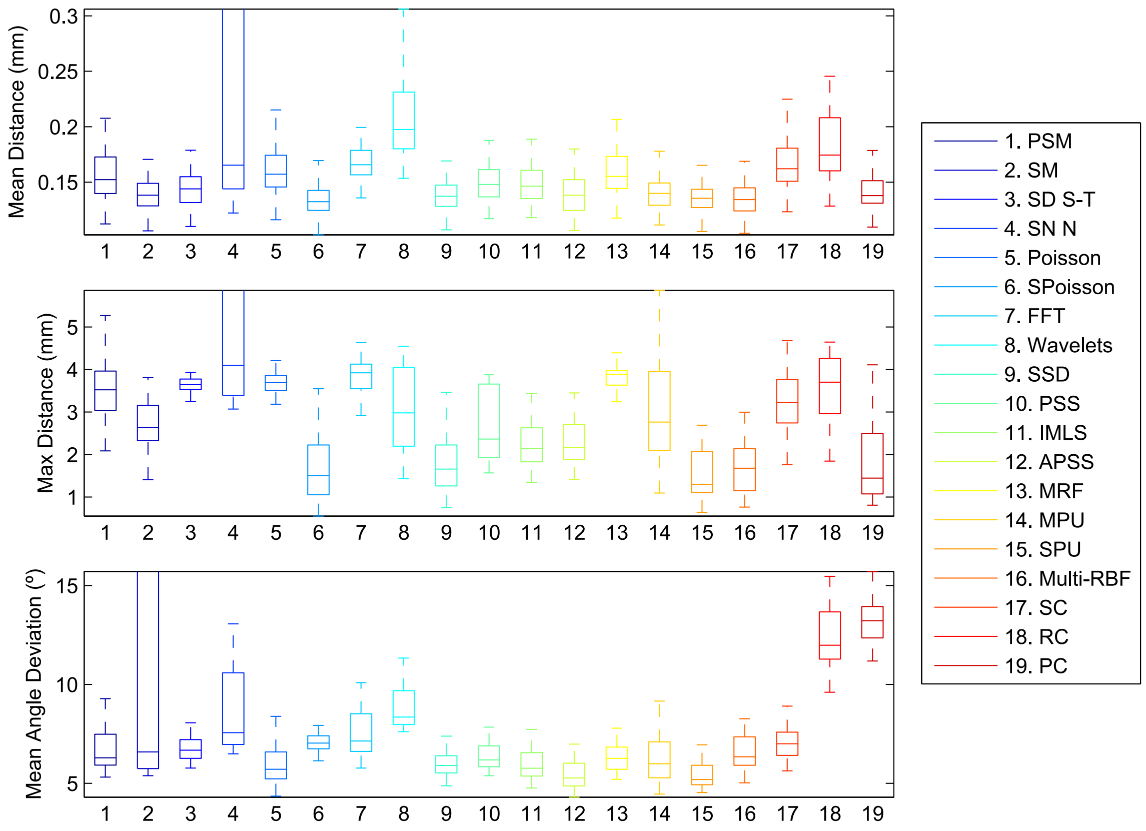


Figure 8.7: Benchmark by Berger et al.[22], results for the **Quasimoto** dataset. We show the mean and maximum (i.e., Hausdorff) distances, and the mean angle deviation for each method when applied to 48 synthetic scans generated using various scanning parameters.

Table 8.1: Table showing additional information on the meshes retrieved for the smooth surfaces experiments in this chapter. The number of connected components (CC), length of boundaries (B) and the binary flag indicating whether the surface is manifold or not (M) are averaged across all the meshes retrieved.

Algorithm	**Gargoyle**			**Dancing Children**			**Quasimoto**		
	CC	**B**	**M**	**CC**	**B**	**M**	**CC**	**B**	**M**
PSM	2298.60	0.8680	0	893.81	7.07	0	322.23	8.14	0
SM	1661.80	181.22	0.15	1525.02	9.34	0	538.69	13.70	0
SD S-T	10.90	126.70	1	41.27	1.25	1	9.10	87.83	1
SD N	1.40	139.69	1	1.38	107.07	1	1.27	1.14	1
Poisson	1.14	0.3044	1	1.10	0.125	1	1.60	0.09	1
SPoisson	1.13	0	1	1.02	0	1	1.06	0	1
FFT	16.9792	0	1	60.65	0	1	8.38	0	1
Wavelets	1.5	0.0402	1	2079.93	0.07	1	3533.77	0.02	1
SSD	2.08	0.35	0.98	2.27	0.60	1	1.44	0.25	1
PSS	73.97	68.60	0.02	131.85	1.59	0	41.62	95.01	0
IMLS	21.5	72.90	0	27.42	1.99	0	8.75	104.98	0
APSS	57.15	99.62	0.04	65.17	1.41	0	26.58	1.05	0
MRF	1.04	0	1	1.60	2.35	1	1	0	1
MPU	3.65	0.32	1	4	11.34	0.97	3.04	0.91	1
SPU	2.04	0.29	0.85	2.21	0.35	0.93	1.37	0.11	1
Multi-RBF	1.15	0	1	1.13	0.04	1	1.13	0	1
SC	2.33	354.97	1	2.08	531.68	1	1.69	1.76	1
RC	89291.93	5.84	0	77476.89	8.77	0	225.83	4.06	0
PC	1.77	0.17	1	1.92	0.16	1	1.94	1	1

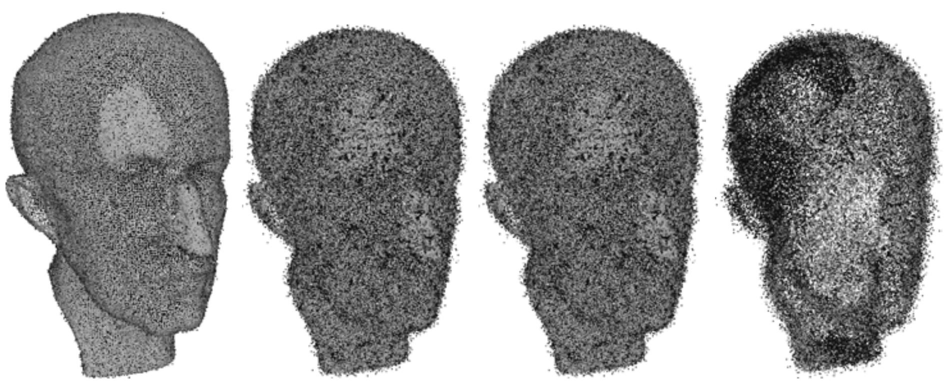


Figure 8.8: Four sample point sets of the Max Planck shape, from a total of 44. Under each figure, we detail the level of noise magnitude (n) and laser's field-of-view size (f). Normals are used to apply shadow casting to the model.

the thickness of the laser beam, with smaller beams being able to detect smaller details. In both cases, these parameters disturb the peak detection process when computing the projection of the laser onto the camera's image plane, thus producing triangulation errors and, consequently, noisy point clouds.

More precisely, we modify the noise magnitude from 0 to 0.5, with increments of 0.05, and the laser's field of view from 2.5 to 10, with increments of 2.5. This gives a total of 44 test point sets. By taking into account the difference between our results and those presented in the article by Berger et al. [22], we can conclude that we are considering a larger amount of noise than in the original comparison (they do not clearly state the values of noise used in their work). We present some samples of noisy point clouds in Figure 8.8.

8.4.3.1 Results

First, we present the results following the approach presented in the original article, i.e., using the error distribution plots previously presented in the above section. Using these plots, depicted in Figure 8.9, one can see the overall behaviour of the methods under varying noise levels.

Regarding mean/max errors, clearly the MPU method is the one obtaining the worst results, and seems unable to handle large amounts of noise. Nevertheless, the SPU method, which is a broad smoothing on the primitives of the original MPU approach, obtains very good estimations. Additionally, the SC, RC and PC methods, all working with raw point sets, do not achieve good results. The gradients-based methods (i.e., Poisson, SPoisson, FFT, Wavelets and SSD), as well as the Multi-RBF, all behave similarly. However, the SSD method attains a larger variability in this case. Additionally, due to its stiffness to the input points, the SPoisson variant reproduces a box a little wider (i.e., more variable errors) than the original Poisson method, since its known smoothing resolves the noise issue better. From the MLS methods, the one behaving the worst is the IMLS. Regarding the methods presented in this thesis, we can see how the PSM and SM methods behave erratically, both obtaining a wide box despite having a low median. On the contrary, the SD S-T and SD N methods behave favorably, achieving the best results in terms of variance in the plots, with SD N obtaining values a bit larger for the Hausdorff distances.

As previously mentioned, all the parameters have been tuned to obtain the best results for each point set. Thus, it is worth mentioning that we used different configurations for the SM, SD S-T and SD N methods for this results. As mentioned in previous chapters, the advantage of the SD S-T and SD N methods over the SM algorithm is their ability to further reduce the inconsistencies in the splat representation (i.e., the noise on the splats). Thus, for the SM case, we tuned the parameters according to each point set. On the contrary, for the SD S-T/N case we obtained the splats required using a fixed set of parameters able to contain the noise level, but also generating inconsistent splats that would greatly disturb the results in the SM case. Thus, the final noise reduction in the SD S-T/N cases is twofold: on the one hand, we have the inherent reduction provided by the splats creation, and on the other hand, the one provided by the global cut on the UDF. This explains the good results obtained by these methods.

If we then focus on the mean angle deviation, our methods, working on raw point sets, SD S-T and SD N obtain a surprisingly good measure, comparable to those of Poisson, Wavelets, PSS, APSS and SPU, all of which work with the additional knowledge of per-point normals. Contrarily, the methods achieving larger variation in this error measure are our other two methods, PSM and SM, along with MPU, SC and PC.

Despite providing a global view of the methods' behaviour, the error distribution plots do not bring information on the amount of noise that a given method is able to assess. For this reason, we additionally show the individual mean and maximum distance for each result, sorted by the increasing noise level. We show a separate chart for each field of view value, and we plot the increasing magnitude of added noise. This allows us to observe under how much noise a given method fails, and also provides more detailed information regarding the method's stability. We provide a macro view of these results in Figure 8.10, where the original values are shown. Then, since details are precluded by large values, we also show a more detailed view of the results in Figure 8.11, by focusing on the smaller values on the plot.

Using these plots, we can clearly see how an increase in noise progressively degrades the results obtained, which is worsened at each incremental step of the laser's field of view. Surprisingly enough, we can see how some specific configurations cause some of

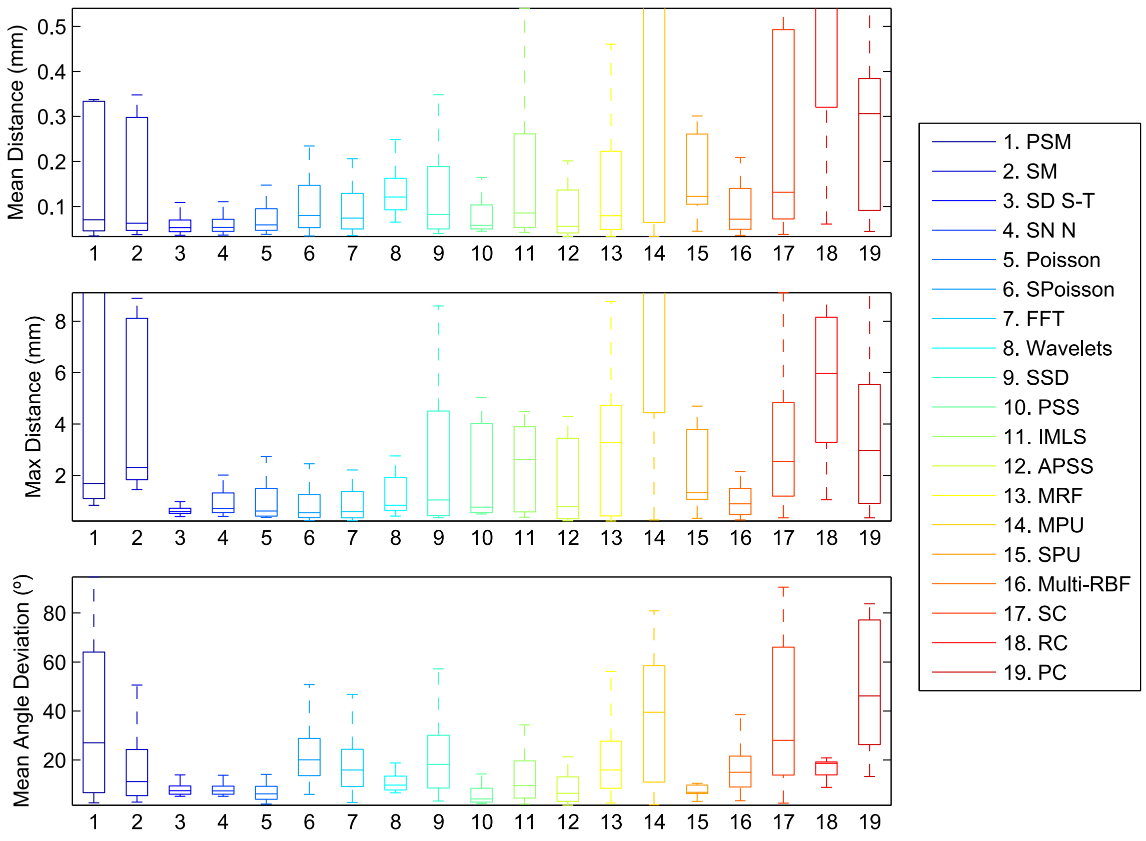


Figure 8.9: Error distribution plots for the noise test of the **Max Planck** dataset, using the benchmark by Berger et al.[22]. We show the mean and maximum (i.e., Hausdorff) distances, and the mean angle deviation for each method when applied to 44 synthetic scans with varying noise scales.

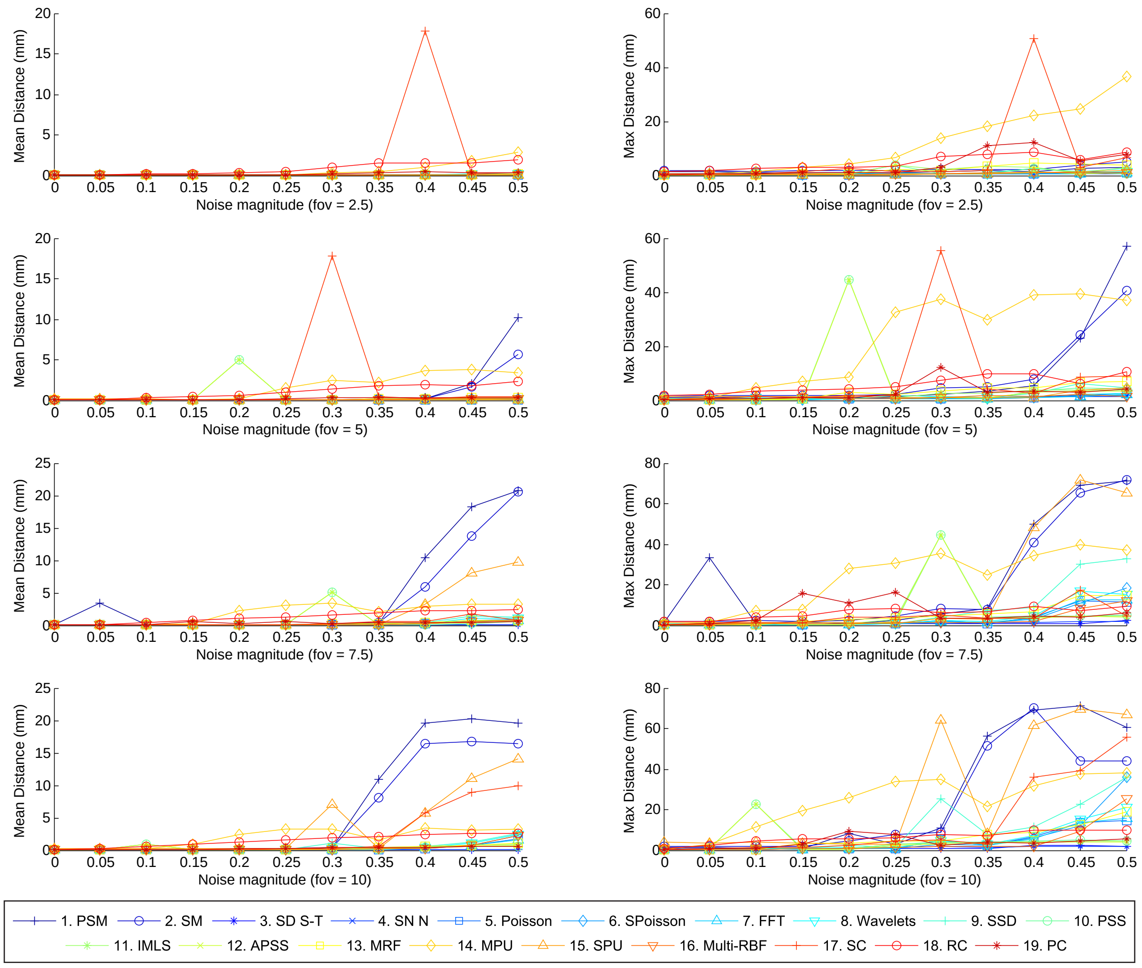


Figure 8.10: Individual results for the noise test on the **Max Planck** dataset. The laser beam's field of view is incremented from top to bottom. Line charts on the left show the mean distance results and the maximum distance on the right.

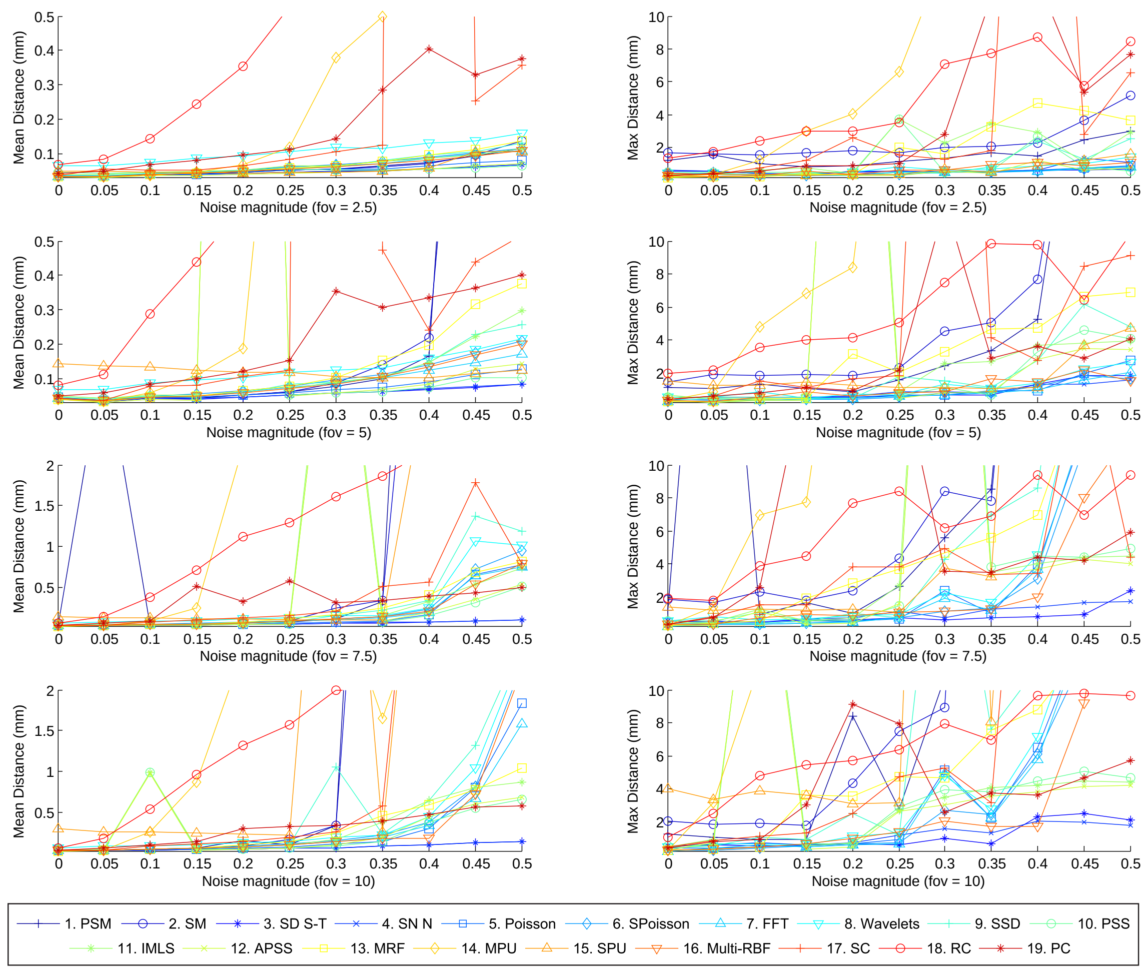


Figure 8.11: Close-up view of the individual results for the noise test on the **Max Planck** dataset. This restricted visualization allows us to observe the details of the results presented. Again, the laser beam's field of view is incremented from top to bottom, and the line charts on the left show the mean distance results, while the ones on the right depict the maximum distance.

the reconstruction methods to fail for a given point set, to then obtain more favorable results in even noisier datasets. This is depicted in the peaks visible for some methods in the line charts. For the smaller laser field-of-view values, the decrease in performance, i.e., the increase in mean/maximum error values, tends to increase steadily for most of the methods (obviously, disregarding the previously mentioned peaks). This raising in the curves step matches the previously mentioned results for the above presented error distribution plots: more variable error distribution plots grow faster in these line charts. Finally, another important point to observe is the noise level due to which some methods totally fail. This is shown with a large step in the error measures.

In the macro view depicted in Figure 8.10, and ignoring the previously mentioned peaks, we can see how the PSM and SM methods have more or less the same resilience to noise. With the exception of the laser's smallest field-of-view value, both methods present a dramatic increase in errors after the same noise level. When the noise scale is too large, and given that these methods are local, the final surface contains non-manifold components that are not included in the reconstruction, thus penalizing the distance score (this can be seen in Table 8.2). We can also see the SPoisson and RC methods having a larger steady increase in both mean/max errors. Also, the SPU method obtains bad results with large noise scales.

Nevertheless, the global view of most of the methods surveyed causes them to be more resilient to noise. Thus, to see the differences between them, we focus on the close-up view of Figure 8.11. In this figure, we can see the methods whose error increases more rapidly. For example, and discarding the previously mentioned methods, the SC, based on raw point sets, is the one with the largest mean/maximum error as the noise increases. However, it is clear that, in terms of noise correction, our SD S-T and SD N algorithms obtain the best results, at the current visualization scale, they are indistinguishable for the means, and the different lines are visible only in the Hausdorff distance charts. The rest of the methods obtain similar curves of progression as the noise increases.

Additional information on the meshes retrieved is again summarized in Table 8.2.

8.4.4 Outliers Test

In this last test, we evaluate the resilience of the state-of-the-art methods against outliers. Note, however, that most of the algorithms tested were not designed to be resilient to outliers. Nevertheless, the global view presented by some of the methods should make them resilient to small amounts of outliers, even if not specifically designed for this problem. Consequently, on the one hand, this evaluation will serve as a demonstration of the poor performance of the current methods in the state of the art to deal with outliers, which is an ubiquitous problem in real data. On the other hand, this evaluation will also quantitatively

Table 8.2: Table showing additional information on the meshes retrieved for the noise experiments in this chapter. The number of connected components (Conn. Comp.), boundaries and the binary flag indicating whether the surface is manifold or not are averaged across all the meshes retrieved.

Algorithm	**Max Planck**		
	Conn. Comp.	**Boundaries**	**Manifold**
PSM	26293.88	2.46	0
SM	331640.65	0	0
SD S-T	1	1	1
SD N	1	1	1
Poisson	1.81	0.28	1
SPoisson	41.5	5.08	1
FFT	99.18	42.07	1
Wavelets	3.32	0.12	1
SSD	830.22	24.45	0.5
PSS	2.02	52.75	0.11
IMLS	1.52	89.23	0.14
APSS	1.27	50.36	0.12
MRF	474.09	73.07	1
MPU	1453.66	127.39	0.30
SPU	428.84	16531.61	0.63
Multi-RBF	41.18	9.28	0.95
SC	1010.57	4543.36	1
RC	5.09	5.76	0.20
PC	47.63	0.21	1

validate the methods' resilience to outliers presented in this thesis, which are specifically required to deal with this type of defect.

Unfortunately, the original code does not provide specific parameterizations to generate outliers in the system. For this reason, we scan the Fertility shape (obtained from the aim@shape repository) using parameters tuned to produce a sufficiently dense point set, with increasing levels of noise magnitude, though we left out the laser's field of view parameter in this case. Then, we corrupt the point sets generated by adding random points, following a uniform distribution, inside a bounding box containing the point set, slightly enlarged by augmenting its diagonal by a 10% in both directions. Also, each of these added outlier points is assigned a random unit vector as its normal. The number of outlier points ranges from 0 to 100% of the outlier-free point cloud count, with an incremental step of 10%.

Obviously, and as previously stated in this thesis, assuming the outliers to follow a uniform distribution may not be realistic. However, it is also the easiest form of outliers: if the points do not follow any structure, they should not be confused with part of the shape. Thus, if the methods are resilient to outliers, they should be able to eliminate these artificially added points.

Again, note that the resilience to outliers for the SD S-T/N case is mainly provided by the splat generation step, and that the results should present only a better performance in noise correction for the SD S-T/N cases with respect to the SM method.

8.4.4.1 Results

We provide the mean and maximum error measurements for each point set in Figure 8.13. We plot the results for increasing levels of outliers for the different levels of noise tested. In this case, we decided not to use the error distribution plots, as the main aim of this test is to find the breaking point for the level of outliers accepted by each method.

When interpreting these results, one has to take into account that the evaluation takes place with the largest connected component on the resulting mesh. This means that, even if the presence of outliers generates small components, the reconstruction results may still be good, as the largest connected component may remain close to the true surface (e.g., see the SPoisson result on Figure 8.12). Additionally, it is difficult to assess when a method has *failed* to reconstruct a shape by just looking at the error measurements. Note that, even when failing, each method produces a different shape after reconstruction. However, since the noise level is fixed for each graphic, the results obtained for the point set with zero outliers should not differ much after the addition of outliers. Thus, the maximum number of outliers a method can handle is usually reflected in the results as a big jump in the error magnitudes, followed by a stabilization or a steady increase.

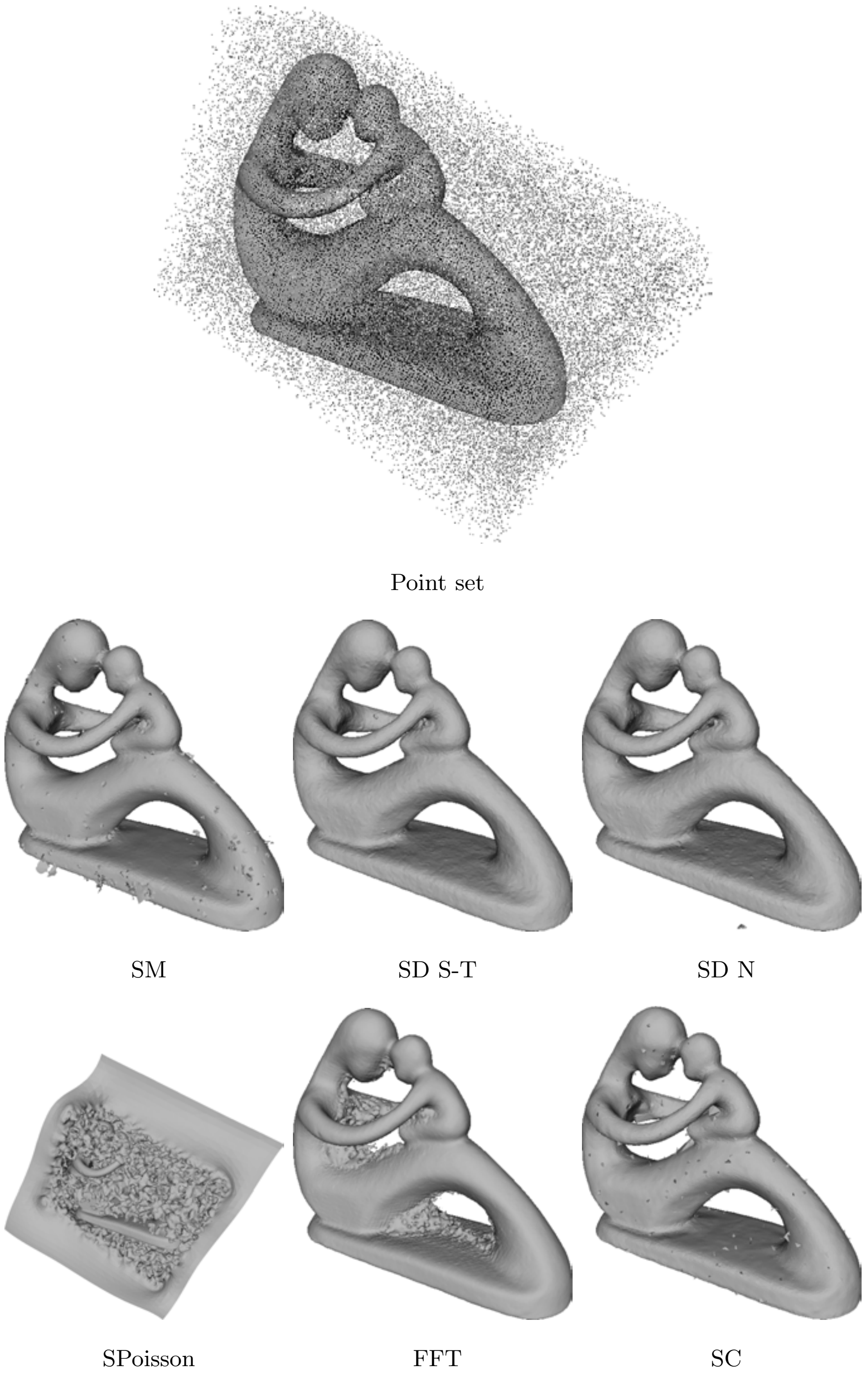


Figure 8.12: Sample results for the outliers case with 50% of outliers. We can see how, even when providing similar results in the comparison, the retrieved surfaces differ qualitatively.

We start by mentioning that the MPU method simply crashes with most of the tested shapes. Moreover, the Wavelets, SSD, PSS, Multi-RBF, RC and PC methods are not even able to handle 10% of outliers. Regardless of the noise level, the Poisson method is able to handle 10% of outliers. Additionally, the IMLS and APSS methods do provide resilience to 10% of outliers for all the noise levels, but they break after that point. The MRF method obtains reasonable results up to 20% of outliers at all but the largest noise level. Also, depending on the point set, the SPU is also able to deal with up to 20% of outliers. The SPoisson method behaves quite well for all the outlier levels without noise, but degrades with increasing noise levels. Finally, FFT and SC perform very well in all cases, although they fail in some specific cases (see the peaks in Figure 8.13). Note that, as expected, the methods presented in this thesis are the ones that perform better under all configurations of both noise and outliers.

In order to stress the qualitative differences in results, even when attaining similar results in the quantitative evaluation, Figure 8.12 shows a view of the shape reconstructed with the methods previously reported to behave correctly for 50% of outliers added. Note the differences in the form of the surfaces retrieved, and their extent, non-manifoldness and missing parts. As in the previous sections, Table 8.3 shows additional information on the presented results, which provides more information regarding the final form of the surface.

Finally, we use Figure 8.14 in order to evaluate the results obtained by our methods. We disregard some peaks for the PSM case for better visualization, but in all cases these peaks are on a small scale, as seen in Figure 8.13. We can observe that the PSM and SM provide the worst results, indicating a lesser noise correction. Additionally, the SD S-T always achieves smaller errors than the SD N case, but they both stay very close to the results obtained for the outlier-free point set for all levels of noise. This proves the added value provided by the methods proposed in this thesis to the outlier rejection problem.

8.5 Conclusions

In this chapter, we provided a quantitative evaluation, not only proving the performance of the methods presented in this thesis, but also of a representative selection of methods from the state of the art.

We started by performing an evaluation of the behaviour of the methods when applied to real datasets arising from different scanning methodologies. This test provides an overall review of the specific performance for the methods against the properties of each point set. Moreover, it serves as a first evidence that the results obtained by the methods presented in this thesis are comparable with those of the state of the art.

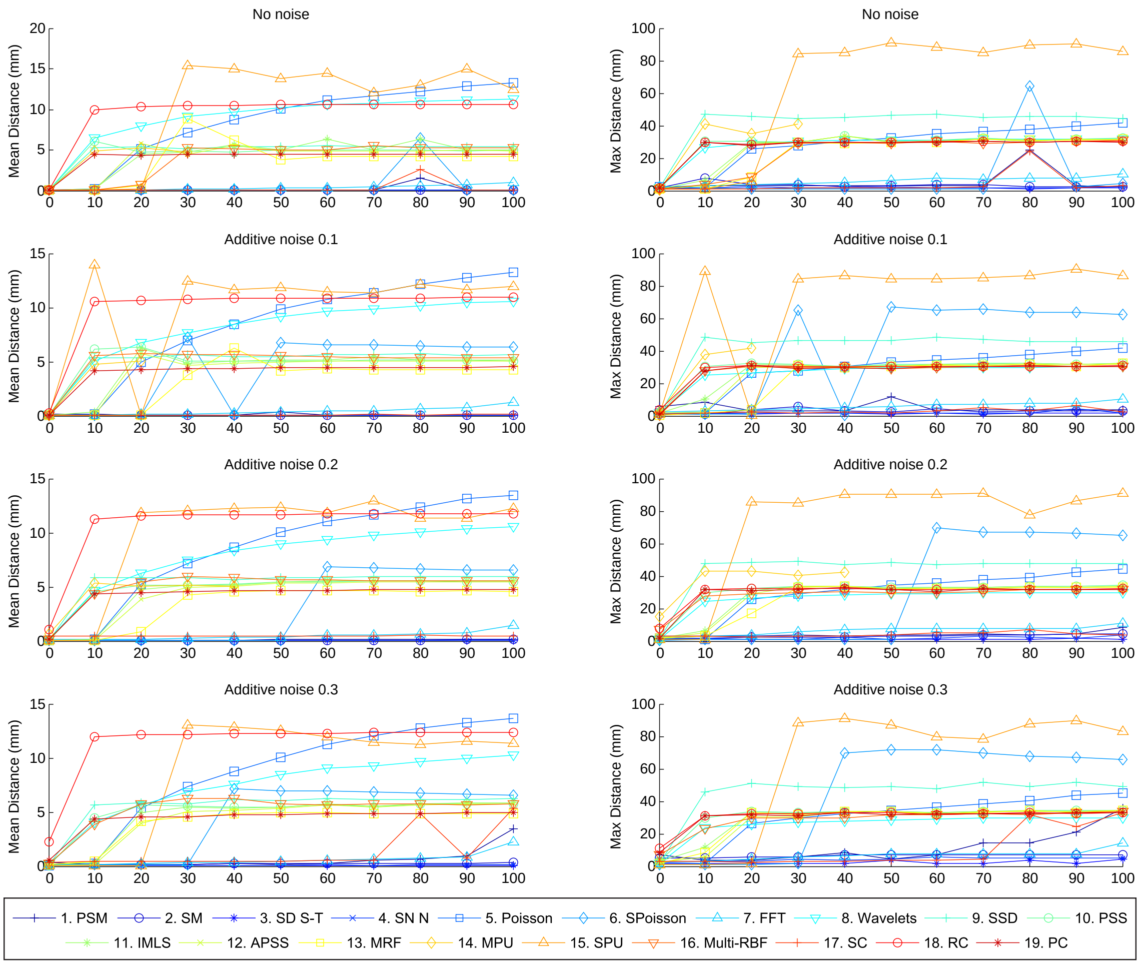


Figure 8.13: Results for the outliers test. Line charts on the left show the mean distance results, and on the right, the maximum (i.e., Hausdorff) distance. Big jumps in the mean/maximum measures clearly point out the break down point of the methods for a given level of noise and outliers.

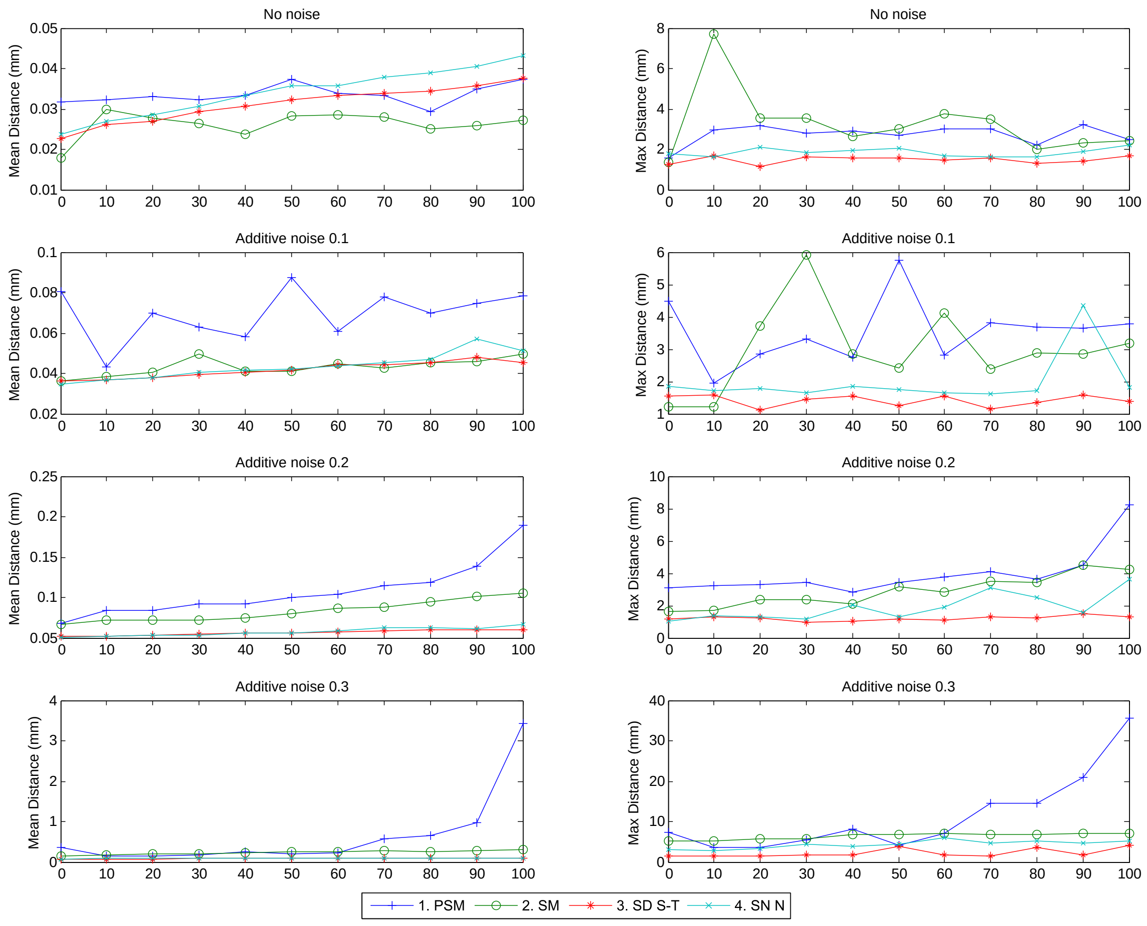


Figure 8.14: Results for the outlier test, showing only the results obtained by the methods proposed in this thesis.

Table 8.3: Table showing additional information on the meshes retrieved for the outliers experiments in this chapter. The number of connected components (Conn. Comp.), boundaries and the binary flag indicating whether the surface is manifold or not are averaged across all the meshes retrieved.

Algorithm	**Fertility**		
	Conn. Comp.	**Boundaries**	**Manifold**
PSM	5517.70	5.03	0
SM	6317.65	0.57	0
SD S-T	1	0	1
SD N	1.22	23.83	1
Poisson	3.86	187.08	0.95
SPoisson	198.27	208.40	1
FFT	401.84	0.30	1
Wavelets	42.38	2.83	1
SSD	3086.22	392.65	0.07
PSS	48.39	128840.26	0.02
IMLS	45.54	252.31	0
APSS	40.95	163.55	0.07
MRF	831.48	6366.20	1
MPU	1931.75	1048.85	0.33
SPU	492	182039.85	0.20
Multi-RBF	314.82	383.21	0.66
SC	53.45	3398.39	1
RC	3.16	10.61	0.27
PC	52.61	0.73	1

The benchmark by Berger et al. [22] was then used to provide a global quantitative positioning of our methods in comparison to the state of the art. We decided to test more methods than in the original reference by including algorithms working with raw point sets. We think that by including these methods we have provided a more complete view of the contributions of the state of the art to the problem of surface reconstruction.

Also, the smooth surfaces test allowed us to state that our methods (PSM, SM, SD S-T and SD N), which work with raw point sets, compare favorably to those in the state of the art requiring normals, which are a clear majority in the review provided in Chapter 4. Nevertheless, the tests revealed a lack of robustness of the parameterizations for the case of SD N, and to a lesser extent SD S-T. Nevertheless, we have proven with the qualitative results on this chapter, that this does not forbid our methods from attaining good results when correctly tuned for a specific dataset.

The noisy tests allowed us to review the robustness to highly corrupted point sets. In concordance with the theory, the approximation-based method, and specially those having a global implicit view of the surface, are the ones that obtain better results. This is in contrast to interpolation-based techniques (SC, RC and PC), which have the worst performance. However, our methods, working with raw point sets, are able to obtain comparable error measures to those methods using oriented point sets as input. Furthermore, the results regarding noise correction obtained by our SD S-T and SD N methods outperform the rest of the methods in the state of the art.

For the outliers test, we found that, even if not specifically designed for this purpose, the FFT and SC methods are surprisingly robust to outliers, even with different noise levels. Also, the outliers' test has proven that our methods are successful in recovering surfaces disregarding these wrong measurements. Using this evaluation, we tried to get a notion of the ability to deal with outliers for the methods of the state of the art. However, we found the error measurements used in other approaches to be quite confusing in this case. These measurements disregard the aberrations of the final shape produced by the further structures that the outliers use to generate. This is due to the use of the largest connected component for the errors. A possible further comparison would be to compute not just the distances between the reference shape and the largest connected component, but also the distances from the outliers to the closest triangle in the reconstructed mesh, including all the components.

Moreover, the variability of user parameters for each method causes the reconstruction results to vary depending on who uses the method and how it is parameterized. It would be desirable for the original authors to test their algorithms against a fixed set of base shapes, so that the best parameters are used, and new methods only have to compare their results against theirs.

To conclude, the work in this chapter has proved that our methods obtain comparable results to those in the state of the art, even when they require an estimation of the normals.

Chapter 9

Conclusions and Future Work

9.1 Summary of the Thesis

The aim of the present thesis was to develop techniques to solve the surface reconstruction problem on highly corrupted point sets, specially on those generated using optical reconstruction techniques in underwater scenarios. We started by reviewing the reconstruction pipeline in the optical case, and the the properties of the resulting point clouds. Then, we surveyed the state of the art in surface reconstruction from unorganized point sets, which threw into sharp relief the lack of approaches working on real non-filtered data, which is likely to contain aberrations such as noise and outliers. With this in mind, we proposed four methods to deal with the problem of surface reconstructions with corrupted data and possibly containing boundaries.

More precisely, in Chapter 4, we have seen how methods in the state of the art can be divided according to whether the surface reconstruction is based on interpolation or approximation. In the first case, the input points or a subset of them are part of the vertices of the output surface. Obviously, if there is noise present in the input data, this will not be corrected by the method, as input points remain unmodified. Nevertheless, a few methods in this category have proven successful in detecting and rejecting outliers. On the other hand, approximation-based techniques deal with the noise attenuation problem implicitly, providing smoother surfaces. However, again few methods try to deal with the outlier rejection problem. Moreover, we should differentiate between methods using raw data (i.e., only the 3D coordinates of the point samples) or additional data, such as per-point normals or the position of the sensor at capture time. In fact, very few proposals work with noisy data without assuming any other kind of additional information associated with the 3D points. Obviously, using additional data eases the construction of robust methods, but restricts their application to their availability. Furthermore, a large majority of methods assume the surface to reconstruct to be closed (i.e., watertight).

After reviewing the state of the art, we decided to provide a generic framework for surface reconstruction by proposing methods that work with raw point sets, but, at the same time, deal with the possible aberrations of noise and outliers that usually appear in real-world datasets. Additionally, all our methods are defined taking into account that the surface may have boundaries, as this case naturally appears when surveying the seafloor. Despite devoting the present thesis to solving the surface reconstruction problem in underwater scenarios, requiring no additional data of any kind makes the proposed methods generic enough to be used with point sets from various types and sources. Accordingly, we proved their application on input data coming from synthetic, range scans, sonar data and computer vision optical reconstruction sources – in this last case, applied to both land and underwater scenes. With respect to our classification of the state of the art, the first two methods proposed in this thesis fall within the *Local Primitives* methods (Section 4.5.8, page 139), while the last two correspond to the *Unsigned Distance* type (Section 4.5.2, page 96).

In Chapter 5, we presented our first proposal based on building on-demand local surfaces to answer intersection queries required by an RDT surface mesher based on Delaunay refinement. In order to mesh an object through RDT surface meshing, the only requirement is to be able to answer a segment intersection query against its surface. In order to provide such a test for raw point sets, given a query segment required by the RDT mesher, we build a local surface using all the points falling at a given radial distance from the query segment. Using the points falling in this capsule-shaped neighborhood, we use a RANSAC procedure to construct the most likely local portion of the surface contained in it in the form of a Local Bivariate Quadric. Additionally, we use an automatic scale selection algorithm to compute the scale of the noise around each point set, and consequently, adapt the RANSAC distance-to-model threshold depending on where the query is made. The method is able to recover bounded surfaces from highly corrupted datasets. However, the computational effort is huge, and recovered surfaces may contain non-manifold structures that may be corrected if needed through a post-processing step.

Then, Chapter 6 proposed an extension over the flaws of the previous method, focusing on diminishing its computational burden by generating an intermediate representation approximating the surface before meshing. We proposed building a splat representation from the point set, where each splat is basically a small surface patch of low degree representing the shape around a given input point. In this way, a set of local splats together in the same frame provides a global approximation of the object. The generation of these splats uses a least squares mechanism which attenuates the noise in the data. Furthermore, and as opposed to the previous approach, we allow the degree of the local approximation to be a user parameter by using jet-surfaces as the parameterized surface the splats are based on. Moreover, these splats are also given an area of influence based on the distance from the generating point to its nearest neighbors. Additionally, in order to ensure using just inlier data in the least-squares minimization, and to reject the outlier points, the jet surfaces are computed as the model of a RANSAC procedure. This intermediate splat representation provides the ability of tuning the quality of the output triangle-based surface at meshing time, providing reusability of the splat representation to obtain multiple resolutions of the same object. In order to mesh the splat representation, we again adapt the RDT mesher by using the splats as a proxy surface to answer the segment intersection queries.

However, the approximation of the object provided by the splat representation is discontinuous and, in order to be meshed through the RDT surface mesher, we provided a mechanism to answer segment intersection queries robustly. We exploit redundancy between splats to answer the intersection test using a 1D RANSAC procedure, so that only intersections supported by two or more splats are reconstructed. Despite still not ensuring the manifoldness of the output meshes, the reconstructed area tends to be larger than in the previous case, and we managed to reduce the computational cost.

Finally, Chapter 7 overviews two proposals that change the local view of previously presented methods into a global formulation. The splat generation procedure has proven successful in removing gross outliers, i.e., those far from the input points. However, we observed that in cases where both double contours and noise are mixed in the data, or when the RANSAC distance-to-model threshold is set too loose, neighboring splats in the representation may become inconsistent with one another. This results in the splats not representing a reasonable approximation of the surface, which is not amenable to the robust intersection detection provided above. Thus, we proposed merging the local contributions of the splats in the above mentioned representation together into a global Unsigned Distance Function (UDF). This UDF is evaluated at the vertices of an adaptive tetrahedral grid which mimics the distribution of the input points, being finer when close to the input data, and coarser otherwise. Furthermore, the function is defined in just a small band around the input points.

Using this distance function, we propose two approaches based on graph cuts to give it a sign, so that the surface can be extracted at its zero level set. The graph structure is inherited from the previous tetrahedralization, and each edge is assigned a weight according to the distance function. Then, the two proposals differ in the partition procedure used.

On the one hand, we use a S-T cut technique, by adding for some vertices in the graph two additional edge links to two virtual nodes `s` (source) and `t` (sink), representing the inside and outside of the shape respectively. In order to compute a confidence for a vertex in the graph to be inside or outside, we again use the approximation provided by the splats.

For each of the vertex needing a labeling cost, we throw rays emanating from the point in random directions and compute the number of intersections with the splats using our robust RANSAC intersection test. Then, given the number of intersections being even or odd, we can tell if the point is inside or outside the shape. Since the ray-splats intersection test may not be perfect, we throw more than one ray and merge the different guesses into one weight to connect the vertex to the `s/t` nodes. Obviously, the method works with the same data as the method in Chapter 6, since it depends on the same intersection procedure. Also, we defined a variation on the method to handle open surfaces within this framework. Basically, we virtually close the surface by also considering for intersection the spherical cap resulting from cutting the bounding sphere of the object with a plane computed by PCA using the centers of the input splats.

On the other hand, the graph can also be partitioned using the Normalized cut method. In this case, the Laplacian matrix, representing the weights in the graph, is used to define a spectral cut in the graph that tries to enforce that edges in the same group have a high weight, edges between different group have a low weight, and that the sum of the costs of the edges in each group have similar weights. In this case, the band in which the distance is defined has to be extended in order to favour the balancing of costs on each partition to cause the cut to pass near the input points. In this case, the partition is only defined in the band, and using this volume partition for meshing, we retrieve a surface of the band of volume. We remove the triangles generated away from the input points by computing the scale of the UDF values at vertices of the mesh near the samples, and remove triangles containing vertices having a too large UDF measure.

We have proven that changing into a global implicit function allows the surface to be coherently defined as the interphase between two volumes in space, ensuring its extraction as a manifold surface, as opposed to previous methods. However, the computational effort and memory requirements increase in comparison to previous proposals.

To conclude this thesis, we applied a quantitative evaluation of the results posed by our methods with respect to those of the state of the art. These results, in addition to the qualitative evaluation performed for each method, allows us to determine that our methods compare favorably to those of the state of the art, while requiring no additional information other than the raw point sets to work.

9.2 Contributions

In the following we highlight the main contributions of this thesis:

- An extensive review of the state of the art on solving the surface reconstruction problem. A global classification of the most relevant methods in the literature is proposed and their main properties are compared and discussed.

- A new method for surface reconstruction, based on building local surfaces on request of the RDT surface mesher. Basically, our main contribution is to transform a pure surface mesher into a surface reconstruction algorithm. By providing the construction of local surfaces on demand, we allow the answering of the intersection queries required by the mesher. Another important improvement over the state of the art is to deal with variable noise scales explicitly. We compute the scale of the noise for each point, and use it to adapt the noise scale during the computation of the local surfaces.

- A two-step method for surface reconstruction, where we decouple the surface approximation and its reconstruction. In the first step, the main contribution is to present the new splat representation, which is a discrete way to approximate the shape of an object using local bounded surfaces as primitives. We propose an innovative use of RANSAC to generate jet surfaces, which allows a remarkable outlier rejection rate while, at the same time, provides noise smoothing. In the second step, the main contribution is to propose a methodology to mesh the new splat representation using RDT Delaunay refinement.

- The proposal of a UDF, merging the local contributions of the splat representation in a global frame. We adapted the implicit MLS definition of Kolluri [124] to allow for LBQs in addition to linear primitives. Furthermore, the UDF is discretized in an adaptive tetrahedral grid, providing rapid evaluation.

- A new method to extract the surface from the UDF, by partitioning the volume using S-T cuts. The main contribution is to define a graph retrieved from the tetrahedralization of the UDF and to partition its vertices into inside/ouside to correctly sign the function. Another remarkable contribution is to use the splat approximation to answer inside/outside queries, and in this way define a confidence for the points in the graph as to being inside or outside the shape.

- A new method to extract the surface from the UDF, by partitioning the volume using Normalized cuts. Using just the smooth weights derived from the UDF, we are able to partition a band of volume by maintaining a similar weight on both sides of the division.

- A qualitative evaluation of the methods under a wide range of input types and sampling conditions presented in the corresponding results section of each chapter.

- A quantitative evaluation of the methods against a wide variety of algorithms in the state of the art. Furthermore, the only benchmark available for surface reconstruction has been extended by including more methods in the evaluation, presenting in this way a global comprehension of the properties of current methodologies, as well as providing a more precise positioning of our proposals with respect to the state of the art.

9.3 Future Work

The work presented in this thesis can be improved and extended in several ways. The following list details future research lines and actions that derive from this thesis:

Reduce the number of splats While we enforced the redundancy of splats in Chapter 6, having a splat for each inlier point may be too complex to define the UDF in Chapter 7. An error based splat reduction scheme may be used at a previous step to reduce the complexity of the splat representation before the creation of the UDF.

Reduce the computational effort When compared to the state of the art methods, our proposals require a far larger amount of computation time. Obviously, we add complexity to the problem, since, as opposed to us, most methods in the state of the art do not assume raw point sets, and do assume an almost clean input. This increase in the complexity of the problematic also translates in an increase in computational effort. Nevertheless, it is worth noting that the implementations of the methods are sequential, and in most cases we can parallelize one or several parts of the algorithms. For instance, the generation of splats is local, and can be easily parallelized once the neighbors of each point are located. Additionally, we would like to explore new data structures that may result in an increase in the performance of the algorithms.

Extension with additional data Even if the methods have been proposed to work with raw point sets, the inclusion of additional data, when available, could help in simplifying some of the steps of the algorithms.

More underwater datasets As mentioned in Section 2.8, the typical mapping techniques for underwater exploration tend to position the scanning sensor in a downward-looking configuration, which prevents the observation of the scene in full 3D. Fortunately, in recent years this has changed to a more general configuration, where the sensor is located at arbitrary positions and with arbitrary orientations in the exploratory vessels. Thus, we expect new datasets to be available in the foreseeable future, so that we can use them to further validate our methods.

Texture mapping Even if we presented some examples using a naive texture mapping approach, it is obvious that the problems arising from using optical imaging in the underwater medium cause the texture mapping problem to require a tailored solution. The most relevant problem in the presented cases is the great change in illumination between different images of the scene, which is clearly visible in Figures 7.18 and 7.19.

Explore further applications The retrieved surfaces are intended to be a tool for further calculus to be applied in this areas. Thus, it would be important to know how geologists, archaeologists or biologists need to manipulate these data, and provide tools for the exploration and further processing of the retrieved surface mesh.

Extension to large scale While the methods shown a successful performance under the presented datasets, the ubiquity and rapid evolution of scanning technologies makes the scanned point sets to become increasingly larger with time. We plan to adapt the presented methodologies to work on these large scale scenarios. A possible idea is to use divide-and-conquer approaches, where the point set is divided into small portions of data, amenable to reconstruction in memory, and then joined together in a global representation.

Release source code It is also important to release the code for the methods presented in this thesis, so the community can benefit from our methods, as well as for providing tools for comparisons in further references to come.

Bibliography

[1] Cgal, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] A. Adamson and M. Alexa. Approximating bounded, non-orientable surfaces from points. In *Shape Modeling International*, pages 243–252. IEEE Computer Society, 2004.

[3] U. Adamy, J. Giesen, and M. John. Surface reconstruction using umbrella filters. *Computational Geometry: Theory and Applications*, 21(1):63–86, January 2002.

[4] M. Alexa and A. Adamson. Interpolatory point set surfaces convexity and hermite data. *ACM Transactions on Graphics*, 28(2):20:1–20:10, May 2009.

[5] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *Proceedings of the Conference on Visualization*, VIS '01, pages 21–28, Washington, DC, USA, 2001. IEEE Computer Society.

[6] M. Alexa, S. Rusinkiewicz, Marc Alexa, and Anders Adamson. On normals and projection operators for surfaces defined by point sets. In *In Eurographics Symposium on Point-Based Graphics*, pages 149–155, 2004.

[7] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, January 2003.

[8] P. Alliez, D. Cohen-Steiner, Y. Tong, and M. Desbrun. Voronoi-based variational reconstruction of unoriented point sets. In *5th Eurographics Symposium on Geometry Processing*, pages 39–48, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[9] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces. In Leila Floriani and Michela Spagnuolo, editors, *Shape Analysis and Structuring*, Mathematics and Visualization, pages 53–82. Springer Berlin Heidelberg, 2008.

[10] N. Amenta and M. Bern. Surface reconstruction by voronoi filtering. In *14th Annual Symposium on Computational Geometry*, SCG '98, pages 39–48, New York, NY, USA, 1998. ACM.

[11] N. Amenta, M. Bern, and D. Eppstein. The crust and the beta-skeleton: Combinatorial curve reconstruction. In *Graphical Models and Image Processing*, pages 125–135, 1998.

[12] N. Amenta, S. Choi, T. K. Dey, and N. Leekha. A simple algorithm for homeomorphic surface reconstruction. In *16th Annual Symposium on Computational Geometry*, SCG '00, pages 213–222, New York, NY, USA, 2000. ACM.

[13] N. Amenta, S. Choi, and R. Kolluri. The power crust. In *6th ACM Symposium on Solid Modeling and Applications*, SMA '01, pages 249–266, New York, NY, USA, 2001. ACM.

[14] N. Amenta and Y. J. Kil. Defining point-set surfaces. *ACM Transactions on Graphics*, 23:264–270, August 2004.

[15] D. Attali. r-regular shape reconstruction from unorganized points. In *13th Annual Symposium on Computational Geometry*, SCG '97, pages 248–253, New York, NY, USA, 1997. ACM.

[16] A. Bab-Hadiashar and D. Suter. Robust segmentation of visual data using ranked unbiased scale estimate. *Robotica*, 17(6):649–660, November 1999.

[17] S. Barkby, S. Williams, O. Pizarro, and M. Jakuba. Bathymetric particle filter SLAM using trajectory maps. *The International Journal of Robotics Research*, 2012.

[18] J. Barnes and P. Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324(6096):446–449, 1986.

[19] T. Barreyre, J. Escartin, R. Garcia, M. Cannat, E. Mittelstaedt, and R. Prados. Structure, temporal evolution, and heat flux estimates from the lucky strike deep-sea hydrothermal field derived from seafloor image mosaics. *Geochemistry, Geophysics and Geosystems*, 13(4):1–29, 2012.

[20] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[21] P.A. Beardsley, A. Zisserman, and D.W. Murray. Navigation using affine structure from motion. In Jan-Olof Eklundh, editor, *Computer Vision - ECCV*, volume 801 of *Lecture Notes in Computer Science*, pages 85–96. Springer Berlin Heidelberg, 1994.

[22] M. Berger, J.A. Levine, L.G. Nonato, G. Taubin, and C.T. Silva. A benchmark for surface reconstruction. *ACM Transactions on Graphics*, 32(2):20:1–20:17, April 2013.

[23] F. Bernardini, J. Mittleman, H. Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, October 1999.

[24] R. Bhotika, D.J. Fleet, and K.N. Kutulakos. A probabilistic theory of occupancy and emptiness. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *European Conference on Computer Vision (ECCV)*, volume 2352 of *Lecture Notes in Computer Science*, pages 112–130. Springer Berlin Heidelberg, 2002.

[25] B. Bingham, B. Foley, H. Singh, R. Camilli, K. Delaporta, R. Eustice, A. Mallios, D. Mindell, C. N. Roman, and D. Sakellariou. Robotic tools for deep water archaeology: Surveying an ancient shipwreck with an autonomous underwater vehicle. *Journal of Field Robotics*, 27(6):702–717, 2010.

[26] S. Bischoff, D. Pavic, and L. Kobbelt. Automatic restoration of polygon models. *ACM Transactions on Graphics*, 24(4):1332–1352, October 2005.

[27] H. Blum. A Transformation for Extracting New Descriptors of Shape. In Weiant W. Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[28] J.-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, October 1984.

[29] J.-D. Boissonnat and F. Cazals. Smooth surface reconstruction via natural neighbour interpolation of distance functions. In *16th Annual Symposium on Computational Geometry*, SCG '00, pages 223–232, New York, NY, USA, 2000. ACM.

[30] J.-D. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67:405–451, September 2005.

[31] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *5th Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 69–78, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[32] R. M. Bolle and B. C. Vemuri. On three-dimensional surface reconstruction methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(1):1–13, 1991. ID: 1.

[33] D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. State of the art in quad meshing. In *Eurographics STARS*, 2012.

[34] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy. *Polygon Mesh Processing.* AK Peters / CRC Press, September 2010.

[35] J.-Y. Bouguet. Calibration toolbox for matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`.

[36] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:359–374, 2001.

[37] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(9):1124–1137, 2004.

[38] Duane C. Brown. Close-range camera calibration. *Photogrammetric Engineering*, 37(8):855–866, 1971.

[39] M. Brown and D.G. Lowe. Unsupervised 3D object recognition and reconstruction in unordered datasets. In *5th International Conference on 3D Digital Imaging and Modeling (3DIM)*, pages 56–63, 2005.

[40] M. Bryson, M. Johnson-Roberson, O. Pizarro, and S.B. Williams. Colour-consistent structure-from-motion models using underwater imagery. In *Robotics: Science and Systems*, 2012.

[41] F. Calakli and G. Taubin. SSD: Smooth signed distance surface reconstruction. *Computer Graphics Forum*, 30(7):1993–2002, 2011.

[42] R. Campos, N. Gracias, R. Prados, and R. Garcia. Merging bathymetric and optical cues for in-detail inspection of an underwater shipwreck. *Instrumentation Viewpoint*, 2013.

[43] J. C. Carr, R. 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 *28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA, 2001. ACM.

[44] F. Cazals and J. Giesen. Delaunay triangulation based surface reconstruction: Ideas and algorithms. In *Effective Computational Geometry for Curves and Surfaces*, pages 231–273. Springer, 2006.

[45] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 177–187, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[46] F. Cazals and M. Pouget. Jet_fitting_3: A Generic C++ Package for Estimating the Differential Properties on Sampled Surfaces via Polynomial Fitting. Research Report RR-6093, INRIA, 2007.

[47] R. Chaine. A geometric convection approach of 3-d reconstruction. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '03, pages 218–229, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[48] M.-C. Chang, F. F. Leymarie, and B. B. Kimia. Surface reconstruction from point clouds by transforming the medial scaffold. *Computer Vision and Image Understanding*, 113(11):1130–1146, NOV 2009.

[49] F. Chazal and A. Lieutier. The λ-medial axis". *Graphical Models*, 67(4):304–331, July 2005.

[50] S.-W. Cheng, T. K. Dey, and E. A. Ramos. Delaunay refinement for piecewise smooth complexes. In *18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1096–1105, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[51] S.-W. Cheng, T. K. Dey, E. A. Ramos, and T. Ray. Sampling and meshing a surface with guaranteed topology and geometry. *SIAM Journal on Computing*, 37(4):1199–1227, November 2007.

[52] L. P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *9th Annual Symposium on Computational Geometry*, SCG '93, pages 274–280, New York, NY, USA, 1993. ACM.

[53] Visual Computing Lab ISTI CNR. Meshlab. http://meshlab.sourceforge.net/.

[54] D. Cohen-Steiner and F. Da. A greedy Delaunay-based surface reconstruction algorithm. *The Visual Computer: International Journal of Computer Graphics archive*, 20(1):4–16, April 2004.

[55] R.T. Collins. A space-sweep approach to true multi-image matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 358–363, 1996.

[56] T. Cour, F. Benezit, and Jianbo Shi. Spectral segmentation with multiscale graph decomposition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 1124–1131 vol. 2, 2005.

[57] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74, 1928. 10.1007/BF01448839.

[58] P. J. Crossno and E. S. Angel. Spiraling edge: Fast surface reconstruction from partially organized sample points. In *10th IEEE Visualization Conference (VIS)*, VISUALIZATION '99, pages –, Washington, DC, USA, 1999. IEEE Computer Society.

[59] G. Cuccuru, E. Gobbetti, F. Marton, R. Pajarola, and R. Pintus. Fast low-memory streaming MLS reconstruction of point-sampled surfaces. In *Graphics Interface*, GI '09, pages 15–22, Toronto, Ont., Canada, Canada, 2009. Canadian Information Processing Society.

[60] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.

[61] B. Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[62] T. K. Dey and J. Giesen. Detecting undersampling in surface reconstruction. In *17th Annual Symposium on Computational Geometry*, SCG '01, pages 257–263, New York, NY, USA, 2001. ACM.

[63] T. K. Dey, J. Giesen, and J. Hudson. Delaunay based shape reconstruction from large data. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, PVG '01, pages 19–27, Piscataway, NJ, USA, 2001. IEEE Press.

[64] T. K. Dey and S. Goswami. Tight cocone: a water-tight surface reconstructor. In *8th ACM Symposium on Solid Modeling and Applications*, SM '03, pages 127–134, New York, NY, USA, 2003. ACM.

[65] T. K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *Computational Geometry Theory and Applications*, 35(1):124–141, August 2006.

[66] T. K. Dey, K. L., E. A. Ramos, and R. Wenger. Isotopic reconstruction of surfaces with boundaries. In *Eurographics/ACM SIGGRAPH symposium on Geometry pro-*

cessing, SGP '09, pages 1371–1382, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[67] T. K. Dey, J. A. Levine, and A. Slatton. Localized Delaunay refinement for sampling and meshing. *Computer Graphics Forum*, 29(5):1723–1732, 2010.

[68] T. K. Dey and J. Sun. An adaptive MLS surface for reconstruction with guarantees. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[69] J. Digne, J.-M. Morel, C.-M. Souzani, and C. Lartigue. Scale space meshing of raw data point sets. *Computer Graphics Forum*, 30(6):1630–1642, 2011.

[70] H. Edelsbrunner. Surface reconstruction by wrapping finite sets in space. *Discrete and Computational Geometry - The Goodman-Pollack Festschrift*, pages 379–404, 2003.

[71] H. Edelsbrunner, M. A. Facello, P. Fu, J. Qian, and D. V. Nekhayev. Wrapping 3D scanning data. In *Proc. SPIE*, volume 3313, pages 148–158, 1998.

[72] H. Edelsbrunner and E. P. Mucke. Three-dimensional alpha shapes. In *Workshop on Volume Visualization*, VVS '92, pages 75–82, New York, NY, USA, 1992. ACM.

[73] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, January 1990.

[74] M. Eigensatz, J. Giesen, and M. Manjunath. The solution path of the slab support vector machine. In *The 20th Canadian Conference on Computational Geometry*, pages 211–214. CCCG, 2008.

[75] A.P. Eriksson, C. Olsson, and F. Kahl. Normalized cuts revisited: A reformulation for segmentation with linear grouping constraints. In *11th IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[76] J. Esteve, P. Brunet, and A. Vinacua. Approximation of a variable density cloud of points by shrinking a discrete membrane. Technical report, Universitat Politècnica de Catalunya, Dept. L.S.I., 2002.

[77] G. E. Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Code.* Academic Press, Inc., Orlando, FL, USA, 4th edition, 1996.

[78] J. Ferrer, A. Elibol, O. Delaunoy, N. Gracias, and R. Garcia. Large-area photomosaics using global alignment and navigation data. In *IEEE Oceans*, pages 1–9, 2007.

[79] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[80] A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *European Conference on Computer Vision*, pages 311–326. Springer-Verlag, 1998.

[81] L. Ford and D. Fulkerson. *Flows in networks.* Princeton University Press, 1962.

[82] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, sep 1977.

[83] S. Fuhrmann and M. Goesele. Fusion of depth maps with multiple scales. *ACM Transactions on Graphics*, 30(6):148:1–148:8, December 2011.

[84] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, August 2010.

[85] R. Gal, Y. Wexler, E. Ofek, H. Hoppe, and D. Cohen-Or. Seamless montage for texturing models. *Computer Graphics Forum*, 29(2):479–486, 2010.

[86] E. Galceran, R. Campos, M. Carreras, and P. Ridao. 3D coverage path planning with realtime replanning for inspection of underwater structures. In *IEEE International Conference on Robotics and Automation (ICRA)*, page to appear, 2014.

[87] R. Garcia, R. Campos, and J. Escartín. High-resolution 3D reconstruction of the seafloor for environmental monitoring and modelling. In *IROS Workshop on Robotics for Environmental Monitoring*, September 2011.

[88] J. Giesen and M. John. Surface reconstruction based on a dynamical system. In *23rd Annual Conference of the European Association for Computer Graphics (Eurographics)*, Computer Graphics Forum 21, pages 363–371, 2002.

[89] J. Giesen and M. John. The flow complex: a data structure for geometric modeling. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03,

pages 285–294, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[90] S. Giraudot, D. Cohen-Steiner, and P. Alliez. Noise-Adaptive Shape Reconstruction from Raw Point Sets. *Computer Graphics Forum*, 32(5):229–238, 2013.

[91] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[92] M. Gopi, S. Krishnan, and C. T. Silva. Surface reconstruction based on lower dimensional localized Delaunay triangulation. *Computer Graphics Forum*, 19(3):C467–+, 2000.

[93] N. Gracias, P. Ridao, R. Garcia, J. Escartin, M. L'Hour, F. Cibecchini, R. Campos, M. Carreras, D. Ribas, N. Palomeras, L. Magi, A. Palomer, T. Nicosevici, R. Prados, R. Hegedus, L. Neumann, F. de Filippo, and A. Mallios. Mapping the moon: Using a lightweight AUV to survey the site of the 17th century ship 'la lune'. In *IEEE Oceans*, June 2013.

[94] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, December 1987.

[95] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(2):271–279, 1989.

[96] G. Guennebaud, M. Germann, and M. Gross. Dynamic sampling and rendering of algebraic point set surfaces. *Computer Graphics Forum*, 27(2):653–662, 2008.

[97] G. Guennebaud and M. Gross. Algebraic point set surfaces. *ACM Trans.Graph.*, 26(3):23.1–23.9, July 2007.

[98] B. I. Guo, J. Menon, and B. Willette. Surface reconstruction using alpha shapes. *Computer Graphics Forum*, 16(4):177–190, OCT 1997.

[99] Vu H. H., R. Keriven, P. Labatut, and J.-P. Pons. Towards high-resolution large-scale multi-view stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1430–1437, 2009.

[100] M. Habbecke and L. Kobbelt. A surface-growing approach to multi-view stereo reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1 –8, june 2007.

[101] G.A. Hansen, R. W Douglass, and A. Zardecki. *Mesh Enhancement: Selected Elliptic Methods, Foundations and Applications.* London: Imperial College Press, 2005.

[102] C. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, 1988.

[103] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[104] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH Computer Graphics*, 26(2):71–78, July 1992.

[105] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 19–26, New York, NY, USA, 1993. ACM.

[106] A. Hornung and L. Kobbelt. Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on dual graph embedding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 503–510, 2006.

[107] A. Hornung and L. Kobbelt. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *4th Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP'06, pages 41–50, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[108] J. Huang and C. H. Menq. Combinatorial manifold mesh reconstruction and optimization from unorganized points with arbitrary topology. *Computer-Aided Design*, 34(2):149–165, 2 2002.

[109] I. Ivrissimtzis, W. Jeong, S. Lee, Y. Lee, and H.-P. Seidel. Surface reconstruction based on neural meshes. In *Mathematical Methods for Curves and Surfaces*, 2004.

[110] I. V. Ivrissimtzis, W.-K. Jeong, and H.-P. Seidel. Using growing cell structures for surface reconstruction. In *Shape Modeling International*, page 78, may 2003.

[111] A. C. Jalba and J. B. T. M. Roerdink. Efficient surface reconstruction using generalized coulomb potentials. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1512–1519, November 2007.

[112] M. Jancosek and T. Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3121–3128, 2011.

[113] M. Jancosek and T. Pajdla. Hallucination-free multi-view stereo. In *11th European Conference on Trends and Topics in Computer Vision*, ECCV'10, pages 184–196, Berlin, Heidelberg, 2012. Springer-Verlag.

[114] M. Johnson-Roberson, O. Pizarro, and S. Willams. Towards large scale optical and acoustic sensor integration for visualization. In *IEEE Oceans - Europe*, pages 1–4, 2009.

[115] M. Johnson-Roberson, O. Pizarro, S. B. Williams, and I. Mahon. Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys. *Journal of Field Robotics*, 27(1):21–51, 2010.

[116] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. *ACM Transactions on Graphics*, 21(3):339–346, July 2002.

[117] M. Kazhdan. Reconstruction of solid models from oriented point sets. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 73–82, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[118] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[119] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Transaction on Graphics*, 32(3):29:1–29:13, July 2013.

[120] M. Kazhdan, A. Klein, K. Dalal, and H. Hoppe. Unconstrained isosurface extraction on arbitrary octrees. In *Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '07, pages 125–133, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[121] T. H. Kim, K. M. Lee, and S. U. Lee. Learning full pairwise affinities for spectral segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2101–2108, 2010.

[122] D.G. Kirkpatrick and J.D. Radke. A framework for computational morphology. In *Computational Geometry, Machine Intelligence and Pattern Recognition*, 1985.

[123] L.P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 57–66, New York, NY, USA, 2001. ACM.

[124] R. Kolluri. Provably good moving least squares. *ACM Transactions on Algorithms*, 4:18:1–18:25, May 2008.

[125] R. Kolluri, J. R. Shewchuk, and J. F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '04, pages 11–21, New York, NY, USA, 2004. ACM.

[126] V. Kolmogorov and R. Zabih. Computing visual correspondence with occlusions using graph cuts. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 508–515 vol.2, 2001.

[127] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *7th European Conference on Computer Vision (ECCV)*, ECCV '02, pages 82–96, London, UK, UK, 2002. Springer-Verlag.

[128] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[129] C.-C. Kuo and H.-T. Yau. A delaunay-based region-growing approach to surface reconstruction from unorganized points. *Computer-Aided Design*, 37(8):825–835, July 2005.

[130] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 307–314 vol.1, 1999.

[131] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(3):277–286, July 2003.

[132] P. Labatut, J.-P. Pons, and R. Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. *2007 Ieee 11th International Conference on Computer Vision, Vols 1-6*, pages 504–511, 2007.

[133] P. Labatut, J. P Pons, and R. Keriven. Robust and efficient surface reconstruction from range data. *Computer Graphics Forum*, 2009.

[134] C. Langmuir, S. Humphris, D. Fornari, C. Van Dover, K. Von Damm, M.K. Tivey, D. Colodner, J.-L. Charlou, D. Desonie, C. Wilson, Y. Fouquet, G. Klinkhammer, and H. Bougault. Hydrothermal vents near a mantle hot spot: the lucky strike vent field at 37Â°n on the mid-atlantic ridge. *Earth and Planetary Science Letters*, 148(1–2):69–91, 1997.

[135] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[136] V. Lempitsky and D. Ivanov. Seamless mosaicing of image-based texture maps. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–6, 2007.

[137] D. Levin. The approximation power of moving least-squares. *Mathematics of Computation*, 67:1517–1531, 1998.

[138] F. F. Leymarie and B. B. Kimia. The medial scaffold of 3D unorganized point clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:313–330, February 2007.

[139] Bao Li, Ruwen Schnabel, Reinhard Klein, Zhiquan Cheng, Gang Dang, and Jin Shiyao. Robust normal estimation for point clouds with sharp features. *Computers & Graphics*, 34(2):94–106, April 2010.

[140] X. Li, C.-Y. Han, and W. G. Wee. On surface reconstruction: A priority driven approach. *Computer-Aided Design*, 41(9):626–640, September 2009.

[141] P. Liepa. Filling holes in meshes. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP'03, pages 200–205, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[142] H.-W. Lin, C.-L. Tai, and G.-J. Wang. A mesh reconstruction algorithm driven by an intrinsic property of a point cloud. *Computer-Aided Design*, 36(1):1–9, 1 2004.

[143] C. A. Lindley. *Practical ray tracing in C.* John Wiley & Sons, Inc., New York, NY, USA, 1992.

[144] S.K. Lodha and R. Franke. Scattered data techniques for surfaces. In *Scientific Visualization Conference*, page 181, june 1997.

[145] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Computer Graphics*, 21:163–169, August 1987.

[146] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions on Mathematical Software*, 36(1):1–30, 2009.

[147] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[148] I. Mahon, O. Pizarro, M. Johnson-Roberson, A. Friedman, S.B. Williams, and J.C. Henderson. Reconstructing pavlopetri: Mapping the world's oldest submerged town using stereo-vision. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2315–2321, 2011.

[149] M. Maila and J. Shi. A random walks view of spectral segmentation. In *AI and STATISTICS (AISTATS)*, 2001.

[150] S. Maji, N. K. Vishnoi, and J. Malik. Biased normalized cuts. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 0:2057–2064, 2011.

[151] J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *International Journal of Computer Vision*, 43(1):7–27, June 2001.

[152] A. Mallios, P. Ridao, D. Ribas, and E. Hernández. Scan matching SLAM in underwater environments. *Autonomous Robots*, 35(1):1–20, 2013.

[153] J. Manson, G. Petrova, and S. Schaefer. Streaming surface reconstruction using wavelets. *Computer Graphics Forum (SGP)*, 27(5):1411–1420, 2008.

[154] S. Marras, F. Ganovelli, P. Cignoni, R. Scateni, and R. Scopigno. Controlled and adaptive mesh zippering. In *International Conference in Computer Graphics Theory and Applications (GRAPP)*, 2010.

[155] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference (BMVC)*, volume 1, pages 384–393, London, 2002.

[156] R. Mencl and H. Müller. Graph-based surface reconstruction using structures in scattered point sets. In *Proceedings of the Computer Graphics International*, CGI '98, page 298, Washington, DC, USA, 1998. IEEE Computer Society.

[157] R. Mencl and H. Müller. Interpolation and approximation of surfaces from three-dimensional scattered data points. In *Dagstuhl '97, Scientific Visualization*, pages 223–232, Washington, DC, USA, 1999. IEEE Computer Society.

[158] N. J. Mitra, A. Nguyen, and L. Guibas. Estimating surface normals in noisy point cloud data. In *Special issue of International Journal of Computational Geometry and Applications*, volume 14(4-5), pages 261–276, 2004.

[159] P. Mullen, F. De Goes, M. Desbrun, D. Cohen-Steiner, and P. Alliez. Signing the unsigned: Robust surface reconstruction from raw pointsets. *Computer Graphics Forum*, 29(5):1733–1741, 2010.

[160] S. Muraki. Volumetric shape description of range data using blobby model. *SIGGRAPH Computer Graphics*, 25(4):227–235, July 1991.

[161] Y. Nagai, Y. Ohtake, and H. Suzuki. Smoothing of partition of unity implicit surfaces for noise robust surface reconstruction. In *Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '09, pages 1339–1348, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[162] T. Nicosevici, N. Gracias, S. Negahdaripour, and R. Garcia. Efficient three-dimensional scene modeling and mosaicing. *Journal of Field Robotics*, 26:759–788, October 2009.

[163] F.S. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, April 2003.

[164] Y. Ohtake, A. Belyaev, and M. Alexa. Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP'05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[165] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Transactions on Graphics*, 22(3):463–470, July 2003.

[166] Y. Ohtake, A. Belyaev, and H.-P. Seidel. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Proceedings of the Shape Modeling International*, page 153, Washington, DC, USA, 2003. IEEE Computer Society.

[167] Y. Ohtake, A. Belyaev, and H.-P. Seidel. 3D scattered data approximation with adaptive compactly supported radial basis functions. In *Proceedings of the Shape Modeling International*, pages 31–39, Washington, DC, USA, 2004. IEEE Computer Society.

[168] Y. Ohtake, A. Belyaev, and H.-P. Seidel. An integrating approach to meshing scattered point data. In *ACM Symposium on Solid and Physical Modeling*, SPM '05, pages 61–69, New York, NY, USA, 2005. ACM.

[169] R. R. Paulsen, J. A. Baerentzen, and R. Larsen. Markov random field surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):636–646, July 2010.

[170] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A pde-based fast local level set method. *Journal of Computational Physics*, 155:410–438, November 1999.

[171] S. Petitjean and E. Boyer. Regular and non-regular point sets: properties and reconstruction. *Computational Geometry*, 19(2-3):101–126, 2001.

[172] L. Piegl and W. Tiller. *The NURBS book*. Springer-Verlag, London, UK, UK, 1995.

[173] A. Rahimi and B. Recht. Clustering with normalized cuts is clustering with a hyperplane. In *Statistical Learning in Computer Vision*, Prague, Czech Republic, 2004.

[174] C. Roman and H. Singh. A self-consistent bathymetric mapping algorithm. *Journal of Field Robotics*, 24(1-2):23–50, 2007.

[175] C. Rother, V. Kolmogorov, and A. Blake. "GrabCut": Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23(3):309–314, August 2004.

[176] P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79:871–880, 1984.

[177] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*. John Wiley & Sons, Inc., New York, NY, USA, 1987.

[178] S. Roy and Ingemar J. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *Sixth International Conference on Computer Vision*, pages 492–499, 1998.

[179] H. Saito and T. Kanade. Shape reconstruction in projective grid space from large number of images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 1999.

[180] N. Salman and M. Yvinec. Surface reconstruction from multi-view stereo. *9th Asian Conference on Computer Vision*, 2009.

[181] M. Samozino, M. Alexa, P. Alliez, and M. Yvinec. Reconstruction with Voronoi centered radial basis functions. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '06, pages 51–60, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[182] S. Schaefer, T. Ju, and J. Warren. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):610–619, May 2007.

[183] S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications*, PG '04, pages 70–76, Washington, DC, USA, 2004. IEEE Computer Society.

[184] O. Schall, A. Belyaev, and H.-P. Seidel. Error-guided adaptive fourier-based surface reconstruction. *Computer Aided Design*, 39:421–426, May 2007.

[185] O. Schall and M. Samozino. Surface from scattered points: A brief survey of recent developments. In Bianca Falcidieno and Nadia Magnenat-Thalmann, editors, *1st International Workshop on Semantic Virtual Environments*, pages 138–147, Villars, Switzerland, 2005. MIRALab.

[186] B. Schölkopf, J. Giesen, and S. Spalinger. Kernel methods for implicit surface modeling. In *Advances in Neural Information Processing Systems 17*, pages 1193–1200. MIT Press, 2005.

[187] S. M. Seitz and C. R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '97, pages 1067–, Washington, DC, USA, 1997. IEEE Computer Society.

[188] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 519–528, 2006.

[189] S. Shalom, A. Shamir, H. Zhang, and D. Cohen-Or. Cone carving for surface reconstruction. *ACM Transactions on Graphics*, 29(6):150:1–150:10, December 2010.

[190] A. Sharf, T. Lewiner, A. Shamir, L. Kobbelt, and D. Cohen-Or. Competing fronts for coarse-to-fine surface reconstruction. In *Computer Graphics Forum*, pages 389–398, 2006.

[191] C. Shen, J. F. O'Brien, and J. R. Shewchuk. Interpolating and approximating implicit surfaces from polygon soup. *ACM Transactions on Graphics*, 23:896–904, August 2004.

[192] J. R. Shewchuk. Constrained delaunay tetrahedralizations and provably good boundary recovery. In *11th International Meshing Roundtable*, pages 193–204, 2002.

[193] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[194] H. Singh, C. Roman, O. Pizarro, R. Eustice, and A. Can. Towards high-resolution imaging from underwater vehicles. *The International Journal of Robotics Research*, 26(1):55–74, 2007.

[195] G.G. Slabaugh, W.B. Culbertson, T. Malzbender, M.R. Stevens, and R.W. Schafer. Methods for volumetric reconstruction of visual scenes. *International Journal of Computer Vision*, 57(3):179–199, 2004.

[196] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, August 2004.

[197] N. Snavely, Steven M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *ACM Transactions on Graphics*, 25:835–846, July 2006.

[198] O. Sorkine and D. Cohen-Or. Least-squares meshes. In *Shape Modeling International*, pages 191–199, Washington, DC, USA, 2004. IEEE Computer Society.

[199] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(4):344–358, April 1995.

[200] F. Steinke, B. Schölkopf, and V. Blanz. Support vector machines for 3D shape processing. *Computer Graphics Forum*, pages 285–294, 2005.

[201] C. Strecha, W. Von Hansen, L. Van Gool, P. Fua, and U. Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[202] T. Svoboda, D. Martinec, and T. Pajdla. A convenient multicamera self-calibration for virtual environments. *Presence: Teleoperators and Virtual Environments.*, 14(4):407–422, August 2005.

[203] G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. In *5th International Conference on Computer Vision*, ICCV '95, pages 902–, Washington, DC, USA, 1995. IEEE Computer Society.

[204] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 311–318, New York, NY, USA, 1994. ACM.

[205] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, 133:97–178, 1908.

[206] C. Walder, O. Chapelle, and B. Schölkopf. Implicit surface modelling as an eigenvalue problem. In *22nd International Conference on Machine Learning*, ICML '05, pages 936–939, New York, NY, USA, 2005. ACM.

[207] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[208] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

[209] R Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–211–I–217 vol.1, 2003.

[210] D. R. Yoerger, D. S. Kelley, and J. R. Delaney. Fine-scale three-dimensional mapping of a deep-sea hydrothermal vent site using the jason rov system. *The International Journal of Robotics Research*, 19(11):1000–1014, 2000.

[211] M. Yoon, I. Ivrissimtzis, and S. Lee. Self-organising maps for implicit surface reconstruction. In *Theory and Practice of Computer Graphics*, pages 83–90, Manchester, UK, June 2008.

[212] Y. Yu. Surface reconstruction from unorganized points using self-organizing neural networks. In *IEEE Visualization*, pages 61–64, 1999.

[213] G. Zeng, S. Paris, L. Quan, and F. Sillion. Progressive surface reconstruction from images using a local prior. In *10th IEEE International Conference on Computer Vision - Volume 2*, ICCV '05, pages 1230–1237, Washington, DC, USA, 2005. IEEE Computer Society.

[214] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[215] H.-K. Zhao, T. Chan, B. Merriman, and S. Osher. A variational level set approach to multiphase motion. *Journal of Computational Physics*, 127:179–195, August 1996.

[216] H.-K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *IEEE Workshop on Variational and Level Set Methods (VLSM)*, VLSM '01, page 194, Washington, DC, USA, 2001. IEEE Computer Society.

[217] H.-K. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit and nonparametric shape reconstruction from unorganized data using a variational level set method. *Computer Vision and Image Understanding*, 80(3):295–314, December 2000.

[218] K. Zhou, M. Gong, X. Huang, and B. Guo. Data-parallel octrees for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, 17(5):669–681, 2011.

[219] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 259–268, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.