Efficient Point-Cloud Processing with Primitive Shapes

Dissertation

zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Dipl.-Inf. Ruwen Schnabel
aus
Konstanz

Bonn, Dezember 2009

Universität Bonn
Institut für Informatik II
Römerstraße 164, D-53117 Bonn

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Reinhard Klein

2. Gutachter: Prof. Dr. Stefan Gumhold

Tag der Promotion: 27.9.2010
Erscheinungsjahr: 2010

CONTENTS

List of Figures **v**

List of Tables **xi**

Abstract **xiii**

Zusammenfassung **xv**

Acknowledgements **xvii**

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Goals . . . 3
- 1.3 Contributions . . . 4
- 1.4 Outline . . . 4
- 1.5 Preliminaries . . . 6
  - 1.5.1 Normals . . . 6
  - 1.5.2 Moving Least-Squares Surface . . . 9
  - 1.5.3 Primitives . . . 13

2 Primitive Detection **17**
- 2.1 Introduction . . . 17
- 2.2 Previous work . . . 18
  - 2.2.1 Vision . . . 18
  - 2.2.2 Reverse engineering . . . 20
  - 2.2.3 Graphics . . . 20
  - 2.2.4 RANSAC . . . 21
- 2.3 Overview . . . 25
- 2.4 Shape estimation . . . 26
- 2.5 Complexity . . . 27
  - 2.5.1 Probabilities . . . 27
- 2.6 Sampling strategy . . . 28
  - 2.6.1 Localized sampling . . . 28

2.6.2 Number of candidates 30
2.7 Score 31
2.7.1 Connected components 32
2.8 Score evaluation 32
2.8.1 Random subsets 32
2.8.2 Octree 34
2.9 Refitting 35
2.10 Out-of-core detection 35
2.10.1 Maximal primitive extent 35
2.11 Alternate score 36
2.11.1 Minimum Description Length 37
2.12 Results 40
2.12.1 Noise 43
2.12.2 Comparison 49
2.12.3 Out-of-core detection 50
2.12.4 Alternate score 51
2.13 Conclusion 53

3 Compression **55**
3.1 Introduction 55
3.2 Previous work 57
3.3 Overview 58
3.4 Compression 60
3.4.1 Resampling 60
3.4.2 Filtering 61
3.4.3 Vector quantization 62
3.4.4 Codebook generation 63
3.4.5 Hierarchy 68
3.4.6 On-disk compression 70
3.5 Decompression 71
3.6 Rendering 72
3.6.1 Level-of-detail 72
3.6.2 Hole-free rendering 72
3.6.3 Normal estimation 73
3.7 Results 74
3.8 Conclusion 78
3.8.1 Limitations and future work 78

4 Recognition **81**
4.1 Introduction 81
4.2 Related work 82
4.2.1 3D city reconstruction 82
4.2.2 Graph-based matching 83
4.2.3 Matching with local features 84
4.3 Overview 84
4.4 Topology Graph 86
4.5 Shape Matching 86
4.5.1 Query graph 87
4.5.2 Constrained subgraph matching 88
4.5.3 First results 90
4.5.4 Query Graph Extensions 90
4.6 Conclusion 95
4.6.1 Future work 96

5 Completion and Reconstruction **97**
5.1 Introduction 97
5.2 Previous work 100
5.3 Shape primitive guided completion 101
5.3.1 Shape primitive detection 103
5.4 Primitive adherence 103
5.4.1 Discrete global minimization 105
5.4.2 Placement of inside and outside constraints 107
5.5 Primitive connectivity 108
5.6 Reconstruction of detail 109
5.7 Surface extraction 110
5.7.1 Consistent edge labeling 111
5.8 Height-fields 112
5.9 Experimental results 114
5.10 Conclusion 119

6 Conclusion **121**
6.1 Discussion 122
6.2 Future work 123

Bibliography **127**

Data Sources **148**

Publications **153**

LIST OF FIGURES

1.1 Several scanned point-clouds from typical application scenarios. From left to right: (a) A scanned street of houses for city planning (courtesy of the Institute for Cartography and Geoinformatics Hannover `http://www.ikg.uni-hannover.de/en/`). (b) Lobby of an office building scanned for building redesign (courtesy of Autodesk Research `http://www.digital210king.org`). (c) A point-cloud of an industrial complex acquired for change management (courtesy of scannTec GmbH & CO. KG, `http://scanntec.com`). (d) A turbine blade for reverse engineering (courtesy of Georgia Tech Large Geometric Models Archive `www.cc.gatech.edu/projects/large_models/`). . . . 2

1.2 Normal estimation: a) and d) The oil-pump and the box model without normals rendered as OpenGL point primitives. b) and e) The estimated normals using an adaptive neighborhood as proposed in Jenke et al. [JKS08]. c) and f) Points colored according to the normal confidence computed as suggested by Pauly et al. [PMG+05] (blue high confidence, red low confidence). Surface edges, registration errors and sampling irregularities reduce the confidence in the normal estimation. 8

1.3 Ray-traced MLS-Surfaces. On the left the point-clouds are shown rendered as simple OpenGL point primitives with normals. Normals are for visualization purposes only and are not required for finding the MLS-Surface. On the right the corresponding ray-traced MLS-Surfaces are shown. The MLS-Surfaces are smooth and contiguous. 11

1.4 The shape primitives considered in this work from left to right: Plane, sphere, cylinder, cone and torus. The coloring in this figure is used throughout this thesis to signify primitive type: red for planes, yellow for spheres, green for cylinders, purple for cones and grey for tori. 13

2.1 A small cylinder that has been detected by our method. The shape consists of 1066 points and was detected among 341,587 points. That corresponds to a relative size of $1/3000$. 29

2.2 To generate this image our algorithm was applied to the barycenters of the triangles. The triangles were than colored according to the shape of their barycenter and the vertices were projected onto the shape. The jagged lines appear because the triangulation does not contain the edges of the shapes. 40

2.3 The chart shows the times of detection of the shapes found in the oil pump model when either subset evaluation or the localized sampling is disabled. For comparison also the timings of the fully optimized version are plotted. Total runtime for the version without subsets was 272.5s, 199.1s without local sampling and 12.3s with both optimizations activated. 41

2.4 The 372 detected shapes in the choir screen define a coarse approximation of the surface. 42

2.5 a) The original scanned model with ca. 500k points. b) Points belonging to shapes in random colors. c) Points of the shapes colored according to the type of the shape they have been assigned to: planes are red, cylinders green, spheres yellow, cones purple and tori are grey. No remaining points are shown. d) The bitmaps constructed for connected component computations provide a rough reconstruction of the object. 44

2.6 a) Distorted model with Gaussian noise and outliers b)-c) Results of the detection on the model with Gaussian noise but without added outliers. d) In addition to the Gaussian noise, 10% outliers were added (see a)). 45

2.7 First column: Original point-clouds. Second column: Shapes colored randomly. Last column: Shapes colored by type as in Fig. 2.5. Models are from top to bottom: rolling stage, house, master cylinder. For parameters and timings see Table 2.1. 46

2.8 First column: Original point-clouds. Second column: Shapes colored randomly. Last column: Shapes colored by type as in Fig. 2.5. Models are from top to bottom: rocker arm, church, and carter. For parameters and timings see Table 2.1. 47

2.9 Points on an octant of a sphere distorted by synthetic gaussian noise with a σ of 10% relative to the sphere diameter and 80% outliers. Our algorithm is able to robustly detect the sphere, see also Table 2.2. 48

2.10 a) A part of the church model containing heavy noise. b) Points belonging to shapes in random colors c) Points colored by type of shape as in Fig. 2.5. 48
2.11 Primitives detected by the out-of-core version of the detection algorithm on 25 Mio. sample points from an aerial stereo reconstruction of Graz. The detection time was 470 sec. 50
2.12 Comparison of results obtained with a simple thresholding score and the MDL based scoring approach. Left column: Input point-clouds. Middle column: Detected primitives with thresholding score, colored by type (cf. Fig. 2.7 and 2.8) Right column: Primitives detected with MDL score. Both approaches use the same distance and normal thresholds, bitmap resolution and minimum support. The weight for the parameters in the MDL score was set to 10. Note that the detected types are more consistent for the MDL score and in general primitives with fewer parameters are preferred. 52

3.1 Michelangelo's St. Matthew rendered interactively at 3.31bpp including normals. 56
3.2 Different stages in our compression algorithm. a) The object is decomposed into parts corresponding to shape primitives. b) Height fields over the primitives are generated to describe fine scale details. c) Laplace pyramids are computed for each height field d) Pyramid levels are encoded with vector quantization. 59
3.3 Kd-tree filtering with missing values. Black points depict vectors without missing values. The black line depicts a vector with a missing value on the x axis. The bounding box B_{max} is shown in blue, the infinite box B_{min} is shown in green dashed lines. The violet cluster center is pruned because its minimal distance to B_{min} is larger than the maximal distance of the orange cluster center to B_{max}. 65
3.4 Two consecutive levels of an image quad-tree. Each quad-tree cell contains x^2 pixels. Below the quad-tree tiles the array for the level is depicted. Each entry stores a pointer to the parent tile, the child relation and the code-vector index. Array entries corresponding to partial tiles, i.e. tiles with incomplete occupancy masks, are sorted to the beginning of the array. 69
3.5 The Ephesos point-cloud with colors after compression. 75
3.6 The PSNR of our method compared to that of Kalaiah et al. [KV05] for the David statue. 76

3.7 Some simple interaction trivially supported in our system. A pipe is highlighted by clicking on it. 77

3.8 Close-up of fine detail on Michelangelo's Atlas. Hole free rendering is achieved with our framebuffer pyramid. On the left decompressed normals are used. On the right normals have been estimated in screen space. 79

3.9 The image on the left has been rendered with normals estimated in screen-space. On the right only shape normals are shown. . . . 79

3.10 The behavior of the normal estimation for different distances from the viewer. In the top row the Atlas model has been rendered with compressed normals on the left. In the middle screen-space aliasing was achieved by splatting the points. On the right normals have been estimated. The bottom row shows a zoomed in view. On the left decompressed normals were used and on the right estimated normals are shown. 80

4.1 Two houses viewed from above that are separated by a narrow alley. Primitive shapes have been detected and are depicted in random colors. a) The topology graph was built with a cell width of 50cm b) The cell width for the construction of the topology graph was set to 2m. Note that the roofs have been connected across the narrow alleyway. In c) and d) we show the resulting topology graphs. In d), the additional edges resulting from large cells are shown in red. 85

4.2 Illustration of the constraints that can be used to detect saddleback roofs: The angle α is constraint to be less than 90 degrees and similar for both planes. The intersection line is required to run parallel to the ground. 87

4.3 A scan of a medieval chapel with Gothic windows containing 4.2M points. The windows were detected by matching the query graph with subgraphs of the topology graph. In a) the original point-cloud is depicted. b) shows the support of the detected shape primitives in random colors. In c) the detected columns are highlighted in green. 91

4.4 A scan of a choir screen consisting of 2M points. The query graph for the columns consisted of a cylinder connected to tori at both ends. In a) the original point-cloud is depicted. b) shows the support of the detected shape primitives in random colors. In c) the detected columns are highlighted in green. 92

4.5 a) Detection of dormers on a roof. The roof plane shown in darker green is a context shape of the dormers. b) The query graph containing a context node. 93

4.6 An L-shaped roof may be hipped on either end. This is best modeled by optional nodes in the query graph. a) A matched L-shaped roof in a stereo reconstruction of a city containing 4M points. b) The query graph used for detection. Optional nodes are shown in grey. 94

4.7 Detection of a stairway in a sampled CAD model of a house. The model was converted to a point-cloud by random sampling of the surface. 94

5.1 Reconstruction of the fandisk model. Orange color signifies completed surface parts. (a) The input point-cloud with holes. (b) The final result. Result without the connectivity enforcement algorithm of Sec. 5.5. The disconnected primitive highlighted in red cuts off part of the model. (d) Close-up views of result without consistent edge labels and final result (see Sec. 5.7) 98

5.2 (a) A hole is indicated by the orange area. (b) Planar primitives in the vicinity are extended. (c) Their intersection defines the completed surface. (d-e) Surface areas approximated by primitives are colored. Black surface parts do not correspond to any primitive. Due to inexact primitive estimation, holes cannot always be closed with primitives only and gaps may need to be filled by other means. (f-g) If primitives are not detected for some reason (e.g. due to noise noise or because the surface cannot be approximated by primitives at all) the algorithm should nonetheless use the information of the available primitives and plausibly connect the synthesized surface to the unapproximated areas. (h-j) The effect of the connectivity constraint. (h) Primitives are color coded. (i) The desired reconstruction contains a completed circle. (j) If connectivity of the primitives is not enforced another reconstruction that cuts off the circle is also possible. 102

5.3 The effect of different configurations on the energy term $-\int_S \mathbf{H}(\langle n|v\rangle)dA$ (see text). 104

5.4 (a) Graph construction and cost assignment: The colored edges intersect a primitive and match its orientation. On these edges $\hat{E}_p$ is set to cancel their area costs. (b) High costs result, if a cut does not follow primitives or fails to match their orientation (flashes mark edges with high costs). 106

5.5 Edge connectivity and edge/primitive correspondence. Cut-edges are depicted in red. (a) The cut-edges connected to edge e are highlighted. (b) Cut-edge f is intersected by both primitives. If h is part of the original support S_1^0 of ψ_1 and the cost of g had previously been increased, it is now reset because it is connected to h, see Sec. 5.5. 108
5.6 Illustration of the different cases for the smoothness term V in eq. (5.8). Left: $V_{e,g}$ is of type (2) and will give the distance between the primitives along edge e, while $V_{g,h}$ is of type (3) and gives the distance between the intersections of g. Right: $V_{e,g}$ is of type (3) and evaluates to zero because the primitives intersect within the cube face. 110
5.7 (a) Height field with missing area. (b) Planar primitives are detected on the ridges (c) Our result (d) Completion result obtained by [JT04] 112
5.8 Completion of the carter model. Orange color signifies completed surface parts. Top row: Our final result with sharp features. Middle row: The input point-cloud with holes. Primitive types are colored as follows: plane/red, sphere/yellow, cylinder/green, cone/purple, grey/torus. Bottom row: The result with the algorithm of [LB07]. . 113
5.9 Completion of the master cylinder. (a)-(d): The final result using the detail reconstruction of Sec. 5.6 (e)-(f): Input point-cloud. Areas corresponding to different primitives rendered in random colors. 115
5.10 Removal of defective and undesired data from an aerial height-field (highlighted in orange). Our algorithm infers missing features such as dormers or walls by propagating surrounding structure represented as primitive shapes. 116
5.11 Reconstruction of the oil-pump from 9 scans. Top: Final result. Bottom from left to right: Input point-cloud. Input point-cloud colored by primitives. Final result. 117
5.12 A preliminary result of detail synthesis on the completed parts of the master cylinder. The image inpainting approach of Komodakis [Kom06] is used to complete the height field over the primitive. . . 118

LIST OF TABLES

2.1 Statistics on processed models. ϵ is given as ratio of maximum bounding box width. Results have been averaged over 5 runs and rounded. 41

2.2 Parameter errors for the fitted spheres under different noise conditions compared to ground truth. All values are given in percent of the sphere diameter. The values are, from left to right, the level of gaussian noise σ, the percentage of outliers o, the mean values μ_r for radius and μ_c for center deviation, and the standard deviations σ_r of the radius and σ_c of the center. 49

2.3 Average percentage of correctly detected regions on the 40 test range images of the Segmentation Comparison Project. The threshold controls the ratio of required overlap between ground truth and machine segmented regions. 49

3.1 Effects of the different acceleration methods when applied to the oil-pump model (see Fig. 3.2). BF denotes brute force nearest neighbor search, S subsampling, F filtering with kd-tree, L the search in the list of l nearest neighbors and E ignores points close to a cluster bisector. L and E are approximating strategies, but bitrates are effected by less than 1%. The required time for construction of the kd-trees for the final F/L/E/S algorithm is 0.6sec (included in the overall timings above). 67

3.2 Compression statistics for various models. T_D gives the time for decomposition in hours and minutes. T_{VQ} is the time for vector quantization. All timings were obtained on an Intel Core 2 Duo with 2GB Ram. Bpp are measured with respect to original points. Disk1 bpp uses on-disk compression using simple adaptive arithmetic coding while disk2 bpp lists results for breadth-first serialized quad-trees. Timings and bitrates in parentheses are for points and normals. For each model six levels-of-detail were used. For Ephesos and Industrial no normals were compressed due to their low quality. 74

3.3 Compression statistics for colors on various models. Bpp gives the bits required for colors only, i.e. without heights or normals. . 75

4.1 Some statistics on test models. *#nodes* gives the number of primitive shapes detected in the point cloud (there is one node per primitive in the topology graph). *#edges* states the number of edges in the topology graph. *top.graph* lists the timings for construction of the topology graph. The last column gives the timings for matching the query graph. 96

5.1 Timings for shape detection in seconds (T_s), number of detected primitives ($|\Phi|$), timings for reconstruction in minutes (T_r), virtual size of volume ($|V|$) 118

ABSTRACT

This thesis presents methods for efficient processing of point-clouds based on primitive shapes. The set of considered simple parametric shapes consists of planes, spheres, cylinders, cones and tori. The algorithms developed in this work are targeted at scenarios in which the occurring surfaces can be well represented by this set of shape primitives which is the case in many man-made environments such as e.g. industrial compounds, cities or building interiors. A primitive subsumes a set of corresponding points in the point-cloud and serves as a proxy for them. Therefore primitives are well suited to directly address the unavoidable oversampling of large point-clouds and lay the foundation for efficient point-cloud processing algorithms.

The first contribution of this thesis is a novel shape primitive detection method that is efficient even on very large and noisy point-clouds. Several applications for the detected primitives are subsequently explored, resulting in a set of novel algorithms for primitive-based point-cloud processing in the areas of compression, recognition and completion. Each of these application directly exploits and benefits from one or more of the detected primitives' properties such as approximation, abstraction, segmentation and continuability.

ZUSAMMENFASSUNG

Die vorliegende Arbeit präsentiert Methoden zur effizienten Verarbeitung von Punktwolken auf Basis von simplen Oberflächenprimitiven. Die betrachtete Menge von parametrischen Primitiven setzt sich aus Ebenen, Kugeln, Zylindern und Tori zusammen. Die Algorithmen die in dieser Arbeit entwickelt werden sind ausgerichtet auf Szenarien in denen die vorkommenden Oberflächen gut durch diese Menge an Primitiven repräsentiert werden können, wie etwa im Falle von durch Menschenhand geschaffenen Umgebungen, z.B. Industrieanlagen, Städte oder Gebäudeinnenräume. Ein Primitiv subsumiert eine Reihe von zugehörigen Punkten aus der Punktwolke und dient als ersatzweiser Repräsentant. Daher sind Primitive gut geeignet die unvermeidbare Überabtastung großer Punktwolken anzugehen und bilden die Grundlage für effiziente Algorithmen zur Punktwolkenverarbeitung.

Der erste Beitrag dieser Arbeit ist eine neuartige Detektionsmethode zur Primitiverkennung die effizient auf sehr großen und verrauschten Punktwolken arbeitet. Mehrere Andwendungen der detektierten Primitive werden nachfolgend untersucht und münden in einer Reihe von neuen Algorithmen für Primitiv-basierte Punktwolkenverarbeitung in den Bereichen Kompression, Erkennung und Vervollständigung. Jede dieser Andwendungen nutzt und profitiert direkt von einer oder mehrerer der Eigenschaften der detektierten Primitive wie Approximation, Abstraktion, Segmentierung oder Fortsetzbarkeit.

ACKNOWLEDGEMENTS

First and foremost, I have to thank my supervisor Prof. Dr. Reinhard Klein, whose energetic enthusiasm for the field has never ceased to be a great source of motivation for me as well as the whole group. Without his inspiration and the many fruitful discussions this work would not have been possible.

I also want to express my gratitude to all my former colleagues in the computer graphics group in Bonn. I always enjoyed the very friendly and cooperative atmosphere that you created. In particular, I thank Roland Wahl, Raoul Wessel, Sebastian Möser, Patrick Degener, Christopher Schwartz and Bao Li for all the effort and long nights spent writing papers. Also, I had a very good time sharing an office with Markus Schlattmann and Gero Müller and I want to thank Roland Ruiters for some wonderful discussions over lunch.

Last but not least I need to thank my wife for her patience during all nights that I spent working on some new idea and almost forgot all about her.

CHAPTER 1

INTRODUCTION

In Computer Graphics the term ’point-cloud’ usually refers to an unordered collection of spatial locations - most often in 3D, but other dimensionality is also common - that can additionally be equipped with a set of attributes such as e.g. color or normal information at the respective locations in space. Generally, point-clouds can represent volumetric data as well as surfaces and are indeed employed in both respects. Common to both approaches is the usually unstructured and irregular sampling of the underlying information. Point-clouds are used in various fields of Computer Graphics and can stem from very different sources. Some of the earliest uses of point-clouds can be attributed to procedural modeling and simulation of natural phenomena such as smoke [CHP+79], clouds [Bli82], fire [Ree83] as well as plants [Ree85]. With the complexity of geometric models rising to a degree where available geometric detail is often much higher than representable on comparatively low resolution screens, point-clouds were also introduced as rendering primitives [RL00, ZPvBG01]. They promise simple resampling techniques to adjust the sampling rate of the geometry to that of the screen and thereby reduce overdraw and aliasing. Point-clouds can also serve as intermediate representations during rendering, e.g. for global illumination as in photon-mapping [JC95] or subsurface scattering [Chr07]. Due to their simple structure and resampling capability, point-clouds have also proven very suitable for interactive geometry modeling [Pau03] as well as animation and physical simulation [Kei06]. However, the recent interest in point-clouds has arguably been driven the most by the advent of cheap and versatile 3D acquisition devices. The most popular acquisition techniques, laser range scanning [LPC+00], structured light scanning [GTK92], shape from shading [HB89] and multi-view stereo [HZ04] in general all produce unstructured 3D point-clouds that are sampled from the acquired surface geometry. This work - while intentionally staying oblivious to the technical details of acquisition - deals with exactly this kind of point-clouds and in the following ’point-cloud’ will always refer to a sampling of a surface in 3D which usually was acquired by one of the above techniques (see Fig. 1.1 for some

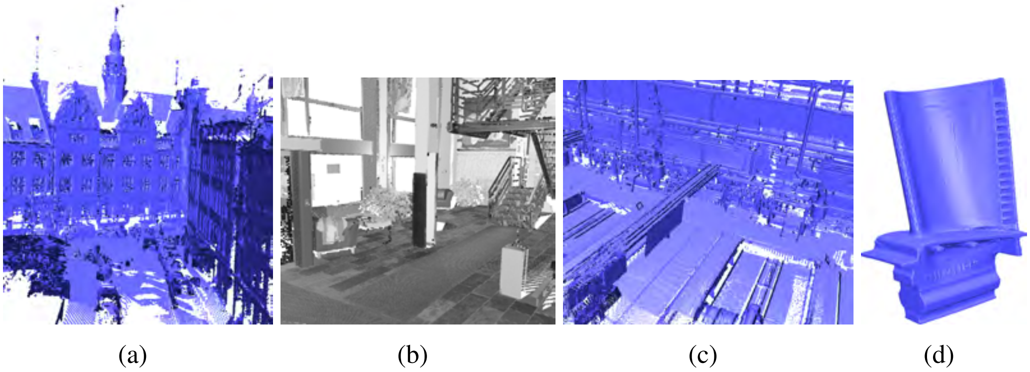

(a) (b) (c) (d)

Figure 1.1: Several scanned point-clouds from typical application scenarios. From left to right: (a) A scanned street of houses for city planning (courtesy of the Institute for Cartography and Geoinformatics Hannover `http://www.ikg.uni-hannover.de/en/`). (b) Lobby of an office building scanned for building redesign (courtesy of Autodesk Research `http://www.digital210king.org`). (c) A point-cloud of an industrial complex acquired for change management (courtesy of scannTec GmbH & CO. KG, `http://scanntec.com`). (d) A turbine blade for reverse engineering (courtesy of Georgia Tech Large Geometric Models Archive `www.cc.gatech.edu/projects/large_models/`).

examples). Such acquired point-clouds find applications in a diverse set of areas: reverse engineering and prototyping, construction and maintenance, quality control, city planning, cultural heritage, forensics, movie production and robotics to mention just a few.

1.1 Motivation

However, point-clouds acquired from real-world objects commonly possess several properties that pose challenges for any algorithm that further processes this raw data. One major issue, which actually does not depend on the acquisition process but rather is a natural property of all point-clouds, is the lack of any structure, or connectivity information respectively, in the point data. This missing information makes it hard to identify the true shape of the underlying surface even in densely sampled regions. In case of real-world scans this difficulty is further intensified by the inevitable presence of noise, which under adverse circumstances can be on an order even larger than the sample spacing. This noise can be caused by measuring principle specific characteristics or environmental influences, e.g. unfavorable lighting conditions, dust or even vibrations. Therefore, algorithms operating on point-clouds need to be robust to this kind of noise as well as outliers. Moreover, acquisition is usually incomplete in the sense that large parts of the geometry remain hidden to the scanning device due to occlusion and restrictions on scanner placement. Often however, it is desired to obtain a complete model of the acquired scene that fills these missing regions - if not necessarily correctly - at least in a plausible manner. Last but not least there is the sheer size of the generated point-clouds which can easily be in the billions. On the one hand, the size of the point-clouds is caused by the amount of acquired geometry of course, e.g. if an entire city is scanned this will necessarily result in a relatively large point-cloud, but on the other hand the size is usually needlessly inflated due to heavy oversampling. This is because the scanning device does not adapt the sampling rate to the acquired geometry and therefore even very flat areas, e.g. the roads and house walls in the city, will be sampled at high rates resulting in an overly redundant point-cloud. It thus seems necessary and reasonable to investigate representations of the data that subsume redundant information on a higher level. In this context, an observation fundamental to this work is that, especially in scenes where man-made objects predominate, as is the case in many of the aforementioned application scenarios, e.g. when scanning mechanical parts, factory sites or other building interiors, large parts of the acquired geometry can usually be well represented by a set of simple parametric shape primitives: planes, spheres, cylinders, cones and tori. Once detected, these primitives achieve the desired effect with respect to the redundancy in the point-cloud, i.e. each primitive subsumes its associated set of corresponding points and can serve as a rough proxy for these. Importantly though, at the same time each primitive remains closely linked to the original data and it is possible to access the subsumed points at any time if operations are to be performed on a more detailed level.

1.2 Goals

The primary goal of this thesis is to address the above mentioned challenges posed by point sampled real-world geometry through the use of these simple primitives. A first step towards this end is developing an efficient method for detection of these primitives. Building on this, the benefit of the primitives in several geometry processing tasks shall be demonstrated. While fitted shape primitives have been used previously for reverse engineering purposes [PLH+05, BMV01] and in computer vision [LGB95, RL93] this work strives to further extend methods in these areas to better exploit the primitives and highlights additional application areas that have not been considered before. In general, the advantages of primitive based algorithms can be expected to be twofold: They should achieve computational efficiency even on large point-clouds due to the concise representation and at the same time - in contrast to general simplification approaches [PGK02, WK04] - offer improved quality due to the geometric and semantic cues provided by the primitives in the specific setting of man-made environments. In particular, applications benefit from the approximating nature of the fitted primitives which also gives important clues to the true shape of the underlying sampled geometry. This can for instance be exploited for compression of the point data. Moreover, the type of a detected primitive can also serve as a very rough classification of the subsumed surface part, which - in conjunction with the segmentation of the surface - is helpful for recognition tasks. Since the primitives are parametric, they also provide local parametrizations of the point-cloud which can for instance be used to infer structured connectivity between the point samples. In addition, these primitives are trivially extensible and can therefore be used to infer missing geometry plausibly in incomplete scans.

1.3 Contributions

In short, the main contributions of this thesis are:

- An efficient and robust algorithm for detection of shape primitives in large real-world point-clouds.

- A compression algorithm based on a primitive-based decomposition of the point-cloud. It supports interactive decompression and rendering on the GPU.

- A feature recognition method based on configurations of shape primitives. It allows automatic extraction of user specified entities.

- An approach to surface completion and reconstruction that uses shape primitives to synthesize missing surface parts.

As usual in the field of computer graphics most of the algorithms presented in this thesis have already been published at several international computer graphics conferences [SWK07, LSSK09, SMK07, SMK08, SWWK08, SDK09].

1.4 Outline

After introducing a novel primitive detection algorithm that operates directly on real-world scanned point-clouds and is specifically designed to handle their size and ubiquitous noise, this work considers several important tasks in point-cloud processing where the above mentioned properties of the detected primitives can be exploited to obtain efficient and effective novel algorithms.

Specifically, Chapter 2 presents and discusses our novel primitive extraction algorithm. The method is a high performance Random Sample Consensus (RANSAC) algorithm [FB81] that is capable to extract the above mentioned types of primitive shapes, while retaining such favorable properties of the RANSAC paradigm as robustness, generality and simplicity. At the heart of our algorithm are a novel, hierarchically structured sampling strategy for candidate shape generation as well as a novel, lazy cost function evaluation scheme, which significantly reduces overall computational cost. The algorithm extracts primitives according to a user specified error tolerance such that even scenes not only comprised of the primitives but also more organic shapes can be decomposed into parts approximated by simple primitives.

Chapter 3 presents a novel point-cloud compression algorithm based on the approximation and parametrization provided by the detected primitives. Geometric detail that is not captured by the primitives is encoded as displacement maps over the primitives. Fast vector quantization is used to compress the displacement maps at several levels-of-detail. The compression is specifically designed to allow fast parallel decompression on the GPU during rendering for interactive applications. The GPU decompression and rendering allows for interactive hole-free level-of-detail point rendering directly from the compressed data. Bitrates of less than four bits per normal-equipped point are achieved. Using only up to two bits per point, high-quality renderings can still be obtained if normals are estimated in image-space.

In Chapter 4 recognition of features in the point-cloud is considered. Building on the segmentation and classification provided by the primitives, features are represented as graphs that describe characteristic configurations in space and orientation of neighbored primitives, e.g. different roof types can be described by graphs capturing the respective characteristic relations between the roof planes. The detection is facilitated by a topology graph that is constructed on all detected primitives and contains edges only between neighboring primitives. Using constrained subgraph matching, features can be extracted from the global topology graph as instances of the given query graphs.

Finally, Chapter 5 presents a completion and reconstruction algorithm that finds a closed mesh using the guidance provided by the detected set of primitive shapes. The extensibility of the primitives is exploited to continue the surrounding structure into the holes and also to synthesize plausible edges and corners from the primitives' intersections. To this end a novel surface energy functional is introduced that incorporates the primitive shapes in a guiding vector field. Finding a surface minimizing the functional gives the desired reconstruction and resolves the possibly occurring ambiguities in a principled manner. The discretized functional can be minimized with an efficient graph-cut algorithm. A novel greedy optimization strategy is proposed to minimize the energy under the constraint that surface parts corresponding to a given primitive must be connected. From the primitive shapes the method can also reconstruct an idealized model that is suitable for use in a CAD system.

1.5 Preliminaries

Before discussing the details of the techniques developed in this work, this section will introduce common notation and provide some background information on point-cloud processing. Throughout this thesis

$$\mathcal{P} = \{(p_1, a_1^1, \ldots, a_1^k), (p_2, a_2^1, \ldots, a_2^k), \ldots, (p_N, a_N^1, \ldots, a_N^k)\}$$

will denote a point-cloud with point coordinates $p_i \in \mathbb{R}^3$ and attributes a_i^j. Typical attributes that will frequently occur are point normals $n_i \in \mathbb{S}^2$ and point colors $c_i \in [0,1]^3$.

1.5.1 Normals

Several of the algorithms presented in this work rely on normal vector information at the point locations. However, scanning devices usually do not provide measurements of surface normals. Therefore, normals usually need to be estimated from the point data in a preprocess.

Finding normals is also the first step in many surface reconstruction methods and a popular approach to normal estimation was first introduced by Hoppe et al. [HDD+92] in this context. The method finds the k-nearest neighbors $\mathcal{N}_k(p) \subset \mathcal{P}$ of a point p and uses the normal to the total least squares best-fitting plane to $\mathcal{N}_k(p)$, i.e. the eigenvector to the smallest eigenvalue of the neighborhood's covariance matrix, as an estimate for the surface normal at p. This approach works well if the points in $\mathcal{P}$ are evenly distributed on the surface, the surface is smooth everywhere and the scale of the noise does not vary much over the surface. The most critical parameter is of course the shape and size of the neighborhood used for fitting the least squares plane. Intuitively, the shape of the neighborhood should be roughly circular in smooth parts of the surface, a larger neighborhood has to be used where the noise is high but conversely a smaller neighborhood should be used in areas of high curvature.

Addressing these partly contradicting requirements, Mitra et al. [MNG04] gave a method for adaptively choosing the size of the neighborhood at different locations in $\mathcal{P}$ based on an analytic approach to bounding the error of the normal under the assumption of a smooth surface. Following intuition, their analytic result strikes a balance between the opposing sources of error caused by noise and curvature. However, their analysis requires a priori knowledge of certain parameters of the sampling distribution as well as reliable estimates of the local curvature and noise variance. In practice these are usually hard (or even impossible) to obtain.

Alternative approaches that do not require these a priori parameters but nonetheless give similar results are based on a multi-scale analysis of the weighted covariance matrix

$$C(\mathcal{N}_k(p)) = \sum_{j \in \mathcal{N}_k(p)} (p_j - \mu_k)(p_j - \mu_k)^T \phi(\|p_j - p\|/h_k), \tag{1.1}$$

where μ_k is the weighted average of all the points in $\mathcal{N}_k(p)$, h_k is the radius of the smallest ball containing $\mathcal{N}_k(p)$ centered at p and ϕ is a positive, monotonously decreasing weighting function, e.g. $\phi(x) = \exp(-x^2)$.

Pauly et al. [PMG⁺05] suggested two local geometry classifiers based on $C(\mathcal{N}_k(p))$ to analyze the distribution of samples in $\mathcal{N}_k(p)$. The first classifier $c_1 \in [0, 1]$ is the normalized weighted least squares error of the estimated tangent plane, i.e. $c_1 = 1 - 3\lambda_0/(\lambda_0 + \lambda_1 + \lambda_2)$ where λ_i are the eigenvalues of $C(\mathcal{N}_k(p))$ and $\lambda_0 \leq \lambda_1 \leq \lambda_2$. The second classifier $c_2 \in [0, 1]$ analyzes the uniformity of the point distribution and is given as $c_2 = \lambda_1/\lambda_2$. The neighborhood size for a point p is then determined by finding a distinct local maximum of the combined classifier $c_1 \cdot c_2$ across the scale axis, i.e. increasing neighborhood sizes.

A similar approach was also proposed by Jenke et al. [JKS08]. The method iteratively increases the size of the neighborhood until the estimated normal and the eigenvalues of $C(\mathcal{N}_k(p))$ stabilize. This has the advantage that potentially less neighborhood sizes have to be considered if stability is detected early on. However, both approaches are heuristics and an error bound cannot be established. Nonetheless we empirically found both methods to yield similar results and to estimate normals with a quality quite sufficient for the algorithms considered in this work (see also Fig. 1.2).

An alternative approach to estimation of surface normals can be derived from the Voronoi diagram of $\mathcal{P}$. If $\mathcal{P}$ is fairly dense, the direction from a point p to the farthest vertex of its Voronoi cell can be used as an estimate of the surface normal at p. This is because the Voronoi cells on the exterior of $\mathcal{P}$ are elongated in a direction perpendicular to the surface. Under specific assumptions about the nature of the noise distribution the error of this estimate can be bounded as demonstrated by Dey and Goswami [DG04]. While in a comparison by Dey et al. [DLS05] the Voronoi based estimation was found to be slightly more accurate than total least squares based approaches, it requires the computation of a Delaunay triangulation on the entire point-cloud $\mathcal{P}$ which can be very time consuming on large input data.

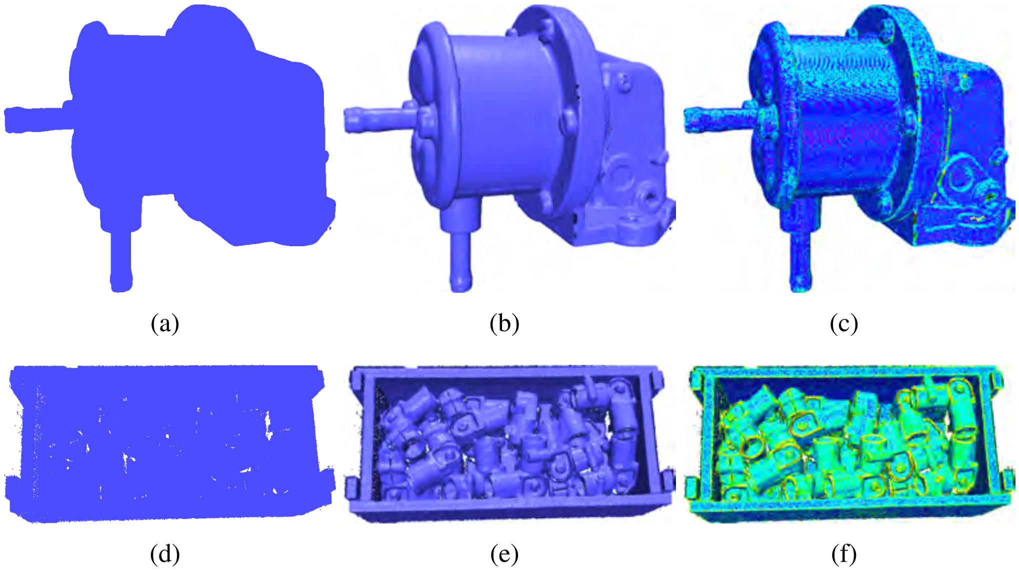


Figure 1.2: Normal estimation: a) and d) The oil-pump and the box model without normals rendered as OpenGL point primitives. b) and e) The estimated normals using an adaptive neighborhood as proposed in Jenke et al. [JKS08]. c) and f) Points colored according to the normal confidence computed as suggested by Pauly et al. [PMG+05] (blue high confidence, red low confidence). Surface edges, registration errors and sampling irregularities reduce the confidence in the normal estimation.

All of the above methods do not explicitly handle sharp features in the point-cloud and therefore generally smooth out edges and corners. To correctly estimate normals close to a sharp feature it is usually sufficient to adjust the shape of the neighborhood $\mathcal{N}(p)$. For instance, for a point close to an edge no points from the opposite side of the edge should be included in $\mathcal{N}(p)$. Several methods based on robust statistics have been proposed to find such adapted neighborhoods [FCOS05, ÖGG09]. The general idea of these approaches is to classify the points on the wrong side of the feature as pseudo-outliers [Ste97]. The definition of sharp features always involves a user defined parameter that controls at which point edges are considered sharp. Moreover, these methods do not address the problem of finding a suitable size of the neighborhood but rely on a user specified global radius.

Therefore, since the algorithms in this work do not require correct normals close to sharp features and can easily deal with some smoothing in these areas, we estimate normals using the method of Jenke et al. which we found to work quite well in practice. If a confidence value for the normal estimation is required, we apply the classifier proposed by Pauly et al. to the neighborhood used for estimating the normal. In principle, Voronoi based approaches might be a viable alternative, but currently suffer from large processing times, especially on large models as are considered in this work.

1.5.2 Moving Least-Squares Surface

At some points in this work it will be necessary to derive a closed surface from the input point-cloud $\mathcal{P}$. Obviously, any reconstruction algorithm, i.e. algorithms creating a triangulated mesh from the input point-cloud, could be used for finding such a surface. However, the reconstructed mesh is only C^1 continuous and - more importantly - full reconstruction is usually an involved and lengthy process that often needs to consider the entire point-cloud $\mathcal{P}$ even if only a small part of the reconstruction will be needed later on. Moreover, a mesh-based representation is comparatively space consuming and not necessarily well suited for the tasks at hand. For instance, implicit surface representations often lend themselves better for computation of ray-surface intersections.

Moving least squares (MLS) surfaces on the other hand were introduced to specifically address these issues and directly use the points themselves as the main source of information about the shape. Therefore MLS surfaces have become the most wide-spread surface definition in the point-based graphics community. MLS surfaces were first introduced by Alexa et al. [ABCO+01, ABCO+03] building on work of David Levin [Lev03]. This first definition of a MLS surface is based on a projection operator: The surface is given as the union of all points stationary under the projection operator. The operator is iterative and at each step a planar approximation of the local neighborhood is found and used as parametrization domain for a subsequent polynomial fitting. The local planar approximation at a point $r \in \mathbb{R}^3$ is defined as the closest local minimum of the following energy:

$$e_{MLS}(\vec{a}, t) = \frac{1}{\alpha} \sum_{p_i \in \mathcal{P}} < \vec{a}, p_i - r + t\vec{a} >^2 \theta(r + t\vec{a}, p_i), \tag{1.2}$$

where $\vec{a}$ is the normal vector of the plane ($\|\vec{a}\| = 1$), $t \in \mathbb{R}$ is the distance from r to the plane, $\theta(x, p_i) = e^{-\|x-p_i\|^2/h^2}$ with a constant scale factor h and $\alpha = \sum_i \theta(r + t\vec{a}, p_i)$ is a normalization constant. The projection of r is given by the intersection of the line $r + \lambda\vec{a}$ with the local polynomial fit. If constant polynomials are used then the projection is simply given as $r + t\vec{a}$. An equivalent formulation for Eq. (1.2) is given by

$$e'_{MLS}(x, \vec{a}) = \frac{1}{\alpha} \sum_{p_i \in \mathcal{P}} < \vec{a}, p_i - x >^2 \theta(x, p_i), \tag{1.3}$$

where $x \in \mathbb{R}^3$. Of course, in order to obtain the same projection as with Eq. (1.2) the minimization must only consider the same three dimensional subspace.

Therefore, finding the correct local plane unfortunately requires expensive non-linear optimization in three dimensions. Amenta and Kil [AK04b] gave a somewhat simplified projection operator that only requires one dimensional non-linear optimization but is still rather involved. Specifically, Amenta and Kil showed that the MLS surface of Eq. (1.2) can be described as a subset of an extremal surface [GM97, MLT00] which in turn is a subset of the zero-level of a certain implicit function. This implicit surface has the following shape:

$$g(x) =< \vec{n}(x), \nabla_y s(y,x)|_x >= 0, \tag{1.4}$$

where $\vec{n}(x)$ is a vector field and $\nabla_y s(y,x)|_x$ is the gradient of an energy $s(y,x)$ (as a function of y) evaluated at x.

Amenta an Kil showed that the MLS surface defined by Eq. (1.3) can in this setting be expressed with the following choices for $\vec{n}(x)$ and $s(y,x)$:

$$\vec{n}(x) = \arg\min_{\vec{a}} e'_{MLS}(x, \vec{a}), \tag{1.5}$$

which is, since fixing x fixes the weights, the normal to the least-squares best fitting plane through x. Furthermore $s(y,x)$ is given as

$$s(y,x) = e'_{MLS}(y, \vec{n}(x)) \tag{1.6}$$

such that $g(x)$ becomes

$$g(x) =< \vec{n}(x), \nabla_y e'_{MLS}(y, \vec{n}(x))|_x > . \tag{1.7}$$

Since the functions $\vec{n}(x)$ and $s(y,x)$ are not fixed, it is possible to overcome the need for non-linear optimization in this setting by replacing the energy $s(y,x)$ with the following slightly modified version:

$$s'(y,x) = \frac{1}{\alpha'} \sum_{p_i \in \mathcal{P}} < \vec{n}'(x), p_i - y >^2 \theta(x, p_i). \tag{1.8}$$

where $\vec{n}'(x) = \arg\min_{\vec{a}} e'_{MLS}(\mu(x), \vec{a})$ with $\mu(x) = 1/\alpha' \sum_{p_i \in \mathcal{P}} p_i \theta(x, p_i)$, i.e. $\vec{n}'(x)$ is the normal to the least-squares best fitting plane to the weighted set $\mathcal{P}$, and $\alpha' = \sum_i \theta(x, p_i)$ is again a normalization constant. Note that in contrast to Eq. (1.6) the weights $\theta(\cdot, p_i)$ no longer depend on y which simplifies the minimization and removes the need for non-linear optimization. The above definitions then give rise to the comparatively simple implicit surface

$$\begin{aligned} g'(x) &= < \vec{n}'(x), \nabla_y s'(y,x)|_x > \\ &= \frac{2}{\alpha'} \sum_{p_i \in \mathcal{P}} < \vec{n}'(x), p_i - x > \theta(x, p_i) \\ &= 2 < \vec{n}'(x), \mu(x) - x > \end{aligned} \tag{1.9}$$

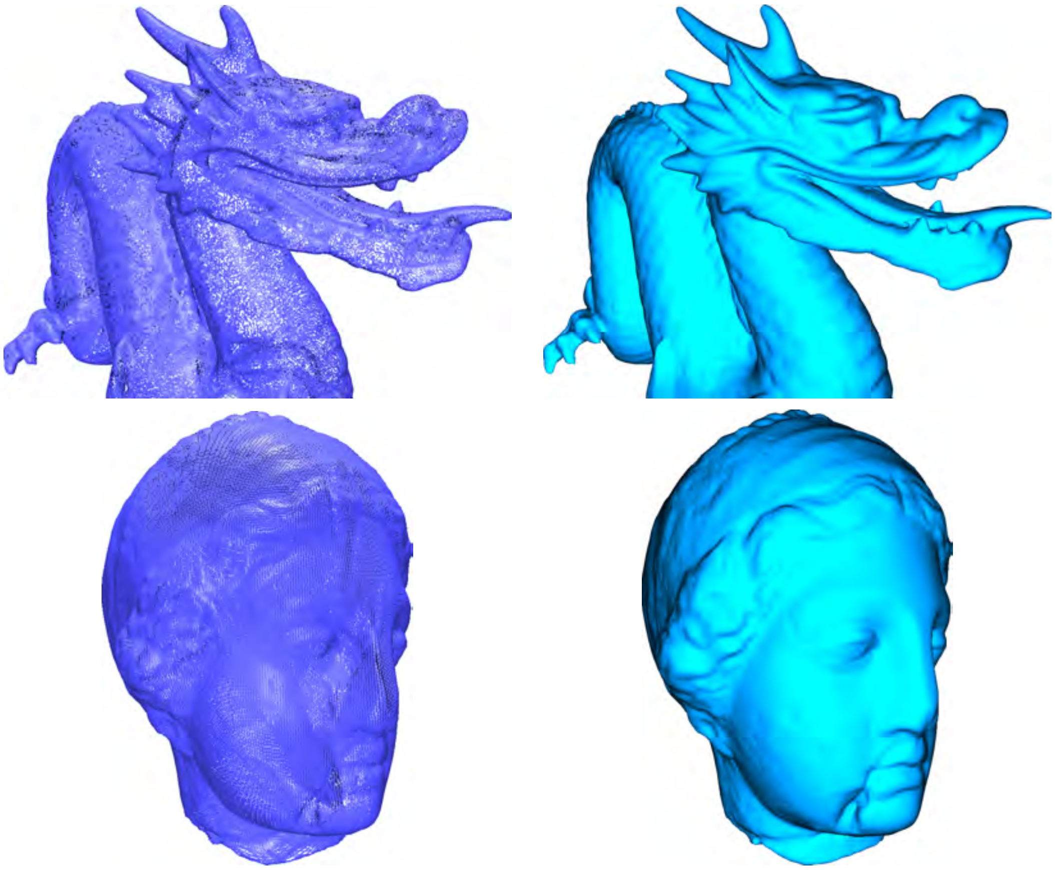


Figure 1.3: Ray-traced MLS-Surfaces. On the left the point-clouds are shown rendered as simple OpenGL point primitives with normals. Normals are for visualization purposes only and are not required for finding the MLS-Surface. On the right the corresponding ray-traced MLS-Surfaces are shown. The MLS-Surfaces are smooth and contiguous.

This surface definition was first introduced by Adamson and Alexa [AA03a, AA04] in the context of ray-tracing of point-clouds (see Fig. 1.3) but was also independently implemented in the PointShop software [ZPKG02, Pau03]. The algorithm for taking a point r to this surface starts by assigning weights $\theta(r, p_i)$ to the points p_i of the input point-cloud $\mathcal{P}$. It finds the total-least-squares best-fit plane H_r to the weighted set $\mathcal{P}$. r is projected onto H_r, giving a new estimate of the projection r', and the method iterates until convergence.

In practice it turns out that the geometrical differences between the surfaces defined by Eq. (1.7) and Eq. (1.9) are negligible for all relevant purposes, but due to the better performance of the linear projection operator Eq. (1.9) is usually the preferred choice and will also be employed in this work. It is however important to note that both surfaces are only defined in a tubular neighborhood near the input point-cloud $\mathcal{P}$ [AK04a]. For practical implementations that means that the above projection procedure cannot be applied to arbitrary points in space without first finding an initial guess that lies within the domain of the surface definition. A common choice in this respect is to use the nearest neighbor to r in $\mathcal{P}$ as the first guess for r'.

Some theoretical guarantees of the reconstructed surface were also proven for several variants of the MLS surface [Kol05, DGS05, DS05] but are based on sampling conditions which are hard or even impossible to achieve in practice.

Another notable variant of the above surface definitions fits algebraic spheres instead of planes and was recently proposed by Guennebaud et al. [GG07, GGG08]. They found that using algebraic spheres increases stability for low-sampling density and also gave some novel operators for real-time upsampling.

The definition as an implicit surface also gives an analytic expression for the normal of the MLS surface [AA04] which, however, is rather complex and therefore is seldom used in practice. Instead, a popular approximation is to use the normal of the local reference plane found in the last iteration of the projection operator. Obviously, this results in normals similar to those described in the previous section.

A problem of the MLS surface as defined above is the inherent low-pass filtering and instability near sharp features. Fleishman et al. [FCOS05] were the first to consider this drawback and proposed the use of robust statistics for feature detection. However their surface definition is not continuous and therefore often produces implausible reconstructions. Öztireli et al. [ÖGG09] recently introduced an alternative approach that is based on robust kernel regression and is able to handle sharp features as well as outliers while always giving a continuous surface.

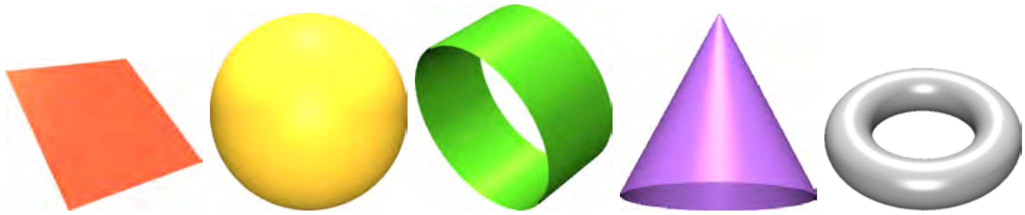


Figure 1.4: The shape primitives considered in this work from left to right: Plane, sphere, cylinder, cone and torus. The coloring in this figure is used throughout this thesis to signify primitive type: red for planes, yellow for spheres, green for cylinders, purple for cones and grey for tori.

1.5.3 Primitives

Simple geometric primitives constitute the major tool of this work and all types of primitives considered in this thesis are depicted in Fig. 1.4. While the primitives employed in this work should be well known to the reader, for completeness, this section will nonetheless give the definitions of the primitives used throughout this thesis. Parametrizations are chosen such that distances in the parameter domain roughly equal distances on the surface.

Plane

The signed distance function for the plane is given by

$$d(x) = < n, x - p > = < n, x > - < n, p >, \tag{1.10}$$

where n, $\|n\| = 1$ is the normal to the plane and p is an arbitrary point in the plane.

A parametric representation of the plane $F : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ is given as

$$F(u, v) = R \begin{pmatrix} u \\ v \\ 0 \end{pmatrix} + p, \tag{1.11}$$

where R is a rotation matrix that specifies the orientation of the plane and maps the z-Axis onto n, i.e. $R = (x|y|n)$ where $x \in \mathbb{R}^3, y \in \mathbb{R}^3, x \times y = n$ are orthogonal vectors in the plane.

Sphere

p being the center of a sphere with radius r the signed distance function is given as

$$d(x) = \|x - p\| - r. \tag{1.12}$$

The following parametrization $F(u,v) : [-r,r] \times [-r\sin(u/r), r\sin(u/r)] \rightarrow \mathbb{R}^3$ of the sphere is used in this work:

$$F(u,v) = r \left(\begin{array}{c} \sin(u/r)\cos(\frac{v}{r\sin(u/r)}) \\ \sin(u/r)\sin(\frac{v}{r\sin(u/r)}) \\ \cos(u/r) \end{array} \right) + p \tag{1.13}$$

Cylinder

The signed distance to a cylinder with an axis $p + \lambda a$, $\lambda \in \mathbb{R}$, $||a|| = 1$ and radius r can be computed as

$$d(x) = \sqrt{||x-p||^2 - < a, x-p >^2} - r. \tag{1.14}$$

The cylinder is parameterized as follows:

$$F(u,v) = R \left(\begin{array}{c} r\sin(u/r) \\ r\cos(u/r) \\ v \end{array} \right) + p, \tag{1.15}$$

where $(u,v) \in [0, 2\pi r] \times \mathbb{R}$ and R is a rotation matrix specifying the orientation of the cylinder, i.e. it maps the z-Axis onto a.

Cone

In this thesis only one-sided cones are considered. For a one sided-cone with apex p and axis $p + \lambda a$, $\lambda \in \mathbb{R} \geq 0$ (a points from the apex into the valid side of the cone), $||a|| = 1$ and semi-angle α the signed distance function is defined as follows:

$$\begin{array}{rcl} g(x) & = & < a, x-p > \\ f(x) & = & \sqrt{||x-p||^2 - g(x)^2} \\ opp(x) & = & g(x) < 0 \wedge f(x) \cdot cos(\alpha) + g(x) \cdot sin(\alpha) < 0 \\ d(x) & = & \left\{ \begin{array}{ll} ||x-p|| & opp(x) \\ f(x) \cdot cos(\alpha) - g(x) \cdot sin(\alpha) & \neg opp(x) \end{array} \right. \end{array} \tag{1.16}$$

We use two different parametrizations for the cone. If $\alpha < 1/4\pi$ we use

$$F(u,v) = R \left(\begin{array}{c} \sin(\alpha)u\sin(v/(sin(\alpha)u)) \\ \sin(\alpha)u\cos(v/(sin(\alpha)u)) \\ \cos(\alpha)u \end{array} \right) + p, \tag{1.17}$$

where $(u, v) \in [0, \infty] \times [-sin(\alpha)u\pi, sin(\alpha)u\pi]$ and R again is a rotation matrix for the orientation of the cone. In case $\alpha \geq 1/4\pi$ we use instead:

$$F(u,v) = R \begin{pmatrix} \sin(\alpha)\sqrt{u^2+v^2}\sin(\operatorname{atan2}(u,v)) \\ \sin(\alpha)\sqrt{u^2+v^2}\cos(\operatorname{atan2}(u,v)) \\ \cos(\alpha)\sqrt{u^2+v^2} \end{pmatrix} + p, \tag{1.18}$$

where $(u, v) \in \mathbb{R}^2$ and $\operatorname{atan2}(y, x)$ gives the angle between the positive x-Axis and the point (x, y).

Torus

The signed distance to a torus is computed as

$$d(x) = \sqrt{g(x)^2 + (f(x) - r_M)^2} - r_m, \tag{1.19}$$

where $g(x)$ and $f(x)$ are defined as above in Eq. (1.16), p is the center of the torus, a is the direction of the torus axis, $\|a\| = 1$, r_M is the major radius and r_m the minor radius.

For $(u, v) \in [-r_m\pi, r_m\pi] \times [-(r_M + \cos(u/r_m)r_m)\pi, (r_M + \cos(u/r_m)r_m)\pi]$ the torus is parameterized as follows:

$$\begin{aligned} \phi_M &= \frac{v}{r_M + \cos(u/r_m)r_m} \\ R_M &= \begin{pmatrix} \cos(\phi_M) & -\sin(\phi_M) & 0 \\ \sin(\phi_M) & \cos(\phi_M) & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ F(u,v) &= RR_M \begin{pmatrix} r_m\cos(u/r_m) + r_M \\ 0 \\ r_m\sin(u/r_m) \end{pmatrix} + p, \end{aligned} \tag{1.20}$$

where R specifies the orientation of the torus, i.e. maps the z-Axis onto a.

CHAPTER 2

PRIMITIVE DETECTION

2.1 Introduction

This chapter presents our novel shape detection method that serves as a prerequisite for all following techniques proposed in this work. Since the shape detection is primarily intended for processing real-world scanned point-clouds it needs to specifically address the challenges of this type of data. In particular, the two most relevant issues arising in this setting are: Firstly, the often huge size of the input data and secondly, the corruption by noise and outliers. A common strategy to address the large size of the point-cloud is to use *local* computations, i.e. consider only those parts of the point-cloud relevant to the current computation. In the case of shape detection we can usually expect a high degree of locality since a single primitive will in general subsume only a small part of the entire dataset. The presence of noise and outliers in the point-cloud on the other hand suggests the use of *robust* techniques for detection and fitting of primitives.

A very popular and versatile robust algorithm is the RANSAC scheme proposed by Fischler and Bolles [FB81] which, beside its simplicity, is very general and effective. Compared to other robust algorithms it has only a relatively small set of parameters which in addition have intuitive interpretations. Moreover, it is able to detect all types of primitives at the same time and concurrently chooses the one of highest merit. Thus, it avoids the need to specify an order in that primitives are considered which otherwise often leads to instabilities and missed primitives. Last but not least, RANSAC requires only very little memory in addition to the point-cloud itself, which is important when working with large datasets.

However, in its original formulation, the RANSAC scheme is a global approach, i.e. it frequently requires operations that consider the entire point-cloud. A RANSAC algorithm creates a large number of candidate shapes by randomly drawing so-called minimal sets from the point-cloud. A minimal-set consists of just enough points to uniquely determine an interpolating primitive. Each candidate's merit is then given as the number of inliers with regard to the respective primitive. Therefore, indeed both major steps of the RANSAC paradigm are global: Samples are drawn globally and determining a candidate's merit considers the entire point-cloud $\mathcal{P}$. Thus, in this chapter we describe an efficient, localized RANSAC algorithm for shape detection in point-clouds:

- We introduce and analyze a hierarchically structured sampling strategy for generation of shape candidates. The sampling strategy accounts for the fact that single primitive shapes constitute only comparatively small and connected subsets of the point-cloud by drawing samples from spatial neighborhoods across a variety of scales.
- The merit of each shape candidate is lazily evaluated on a series of subsets of $\mathcal{P}$. The first subsets of this series are very small and allow to identify and discard candidates of low merit at an early stage. Subsequent subsets become larger but are only evaluated if a candidate has proven to be promising.

Our method detects planes, spheres, cylinders, cones and tori. However, to be feasible, the algorithm requires pre-computed normals associated with the input points (see Sec. 1.5.1). Because normals are usually computed for visualization purposes anyway, this turns out to be a minor limitation in practice though.

2.2 Previous work

The detection of primitive shapes is a common problem encountered in many areas of geometry related computer science. Over the years a vast number of methods have been proposed which cannot all be discussed here in depth. Instead, here we give a short overview of the most important algorithms developed in the different fields. We treat the previous work on RANSAC algorithms separately in section 2.2.4 as it is of special relevance to our work.

2.2.1 Vision

In computer vision, the two most widely known methodologies for shape extraction are the RANSAC paradigm [FB81] and the Hough transform [Hou62]. Both have been proven to successfully detect shapes in 2D as well as 3D. RANSAC and the Hough transform are reliable even in the presence of a high proportion of outliers, but lack of efficiency or high memory consumption remains their major drawback [IK88]. For both schemes, many acceleration techniques have been proposed, but no one on its own, or combinations thereof, have been shown to be able to provide an algorithm as efficient as ours for the 3D primitive shape extraction problem.

The Hough transform maps, for a given type of parameterized primitive, every point in the data to a manifold in the parameter space. The manifold describes all possible variants of the primitive that contain the original point, i.e. in practice each point casts votes for many cells in a discretized parameter space. Shapes are extracted by selecting those parameter vectors that have received a significant amount of votes. If the parameter space is discretized naively using a simple grid, the memory requirements quickly become prohibitive even for primitives with a moderate number of parameters, such as, for instance, cones. Although several methods have been suggested to alleviate this problem [IK87][XO93] its major application area remains the 2D domain where the number of parameters typically is quite small. A notable exception is the method of Vosselman et al. [VGSR04] where the Hough transform is used to detect planes in 3D datasets, as 3D planes still have only a small number of parameters. They also propose a two-step procedure for the Hough-based detection of cylinders that uses estimated normals in the data points.

In the vision community many approaches have been proposed for segmentation of range images with primitive shapes. These algorithms are often based on region growing or region merging. A drawback of this approach is with respect to robustness because primitives need to be initialized from a very small seed region and consequently noise and outliers have a strong adverse influence on the initial estimation. Therefore primitives are usually initialized as planes and several thresholds have to be defined that control when other types of primitives are considered [FEF97, GBS03]. This is a fundamental difference to our approach where points spread far apart can be used to robustly initialize a primitive and no order is imposed on the different types of primitives. Leonardis et al. [LGB95, LJS97] alleviated these issues by concurrently growing different seed primitives and then selecting a suitable subset according to a minimal description length (MDL) criterion (coined the recover-and-select paradigm). However, primitives are still initialized from small patches and the selection procedure is computationally expensive. Gotardo et al. [GBS03] detect shapes using a genetic algorithm to optimize a robust MSAC fitness function (see also Sec. 2.2.4), but this approach is not feasible on large point-clouds. Marshall et al. [MLM01] introduce involved non-linear fitting functions for primitive shapes that are able to better handle geometric degeneracy in the context of recover-and-select segmentation but are still not as robust as our method.

Another robust method frequently employed in the vision community is the tensor voting framework [MLT00] which has been applied to successfully reconstruct surface geometry from extremely cluttered scenes. While tensor voting can compete with RANSAC in terms of robustness, it is, however, inherently model-free and therefore cannot be applied to the detection of predefined types of primitive shapes.

2.2.2 Reverse engineering

In reverse engineering, surface recovery techniques are usually based on either a separate segmentation step or on a variety of region growing algorithms [VMC97, SB95, BGV$^+$02]. Most methods call for some kind of connectivity information and are not well equipped to deal with a large amount of outliers [VMC97]. Also these approaches try to find a shape proxy for every part of the processed surface with the intent of loading the reconstructed geometry information into a CAD application. Benko et al. [BMV01] describe a system which reconstructs a boundary representation that can be imported into a CAD application from an unorganized point-cloud. However, their method is based on finding a triangulation for the point-set, whereas the method presented in this work is able to operate directly on the input points. This is advantageous as computing a suitable tesselation may be extremely costly and becomes very intricate or even ill-defined when there is heavy noise in the data. This chapter, however, does not intend to present a method implementing all stages of a typical reverse engineering process.

2.2.3 Graphics

In computer graphics, Cohen-Steiner et al. [CSAD04] have proposed a general variational framework for approximation of surfaces by planes, which was later extended to a set of more elaborate shape proxies by Wu and Kobbelt [WK05]. Their aim is not only to extract certain shapes in the data, but to find a globally optimal representation of the object by a given number of primitives. However, these methods require connectivity information and are, due to their exclusive use of least squares fitting, susceptible to errors induced by outliers. Also, the optimization procedure is computationally expensive, which makes the method less suitable for large data sets. The output of our algorithm, however, could be used to initialize the set of shape proxies used by these methods, potentially accelerating the convergence of the optimization procedure.

While the Hough transform and the RANSAC paradigm have been mainly used in computer vision, some applications have also been proposed in the computer graphics community. Décoret et al. [DDSD03] employ the Hough transform to identify planes for billboard clouds on triangle data. They propose an extension of the standard Hough transform to include a compactness criterion, but due to the high computational demand of the Hough transform, the method exhibits poor runtime performance on large or complex geometry.

Wahl et al. [WGK05] proposed a RANSAC-based plane detection method for hybrid rendering of point clouds. To facilitate an efficient plane detection, planes are detected only in the cells of a hierarchical space decomposition and therefore what is essentially one plane on the surface is approximated by several planar patches. While this is acceptable for their hybrid rendering technique, our method finds maximal surface patches in order to yield a more concise representation of the object. Moreover, higher order primitives are not considered in their approach.

Gelfand and Guibas [GG04] detect so-called slippable shapes which is a superset of the shapes recognized by our method. They use the eigenvalues of a symmetric matrix derived from the points and their normals to determine the slippability of a point-set. Their detection is a bottom-up approach that merges small initial slippable surfaces to obtain a global decomposition of the model. However, the computation of the eigenvalues is costly for large models, the method is sensitive to noise and it is hard to determine the correct size of the initial surface patches. A related approach is taken by Hofer et al. [HOP$^+$05]. They also use the eigenvalues of a matrix derived from line element geometry to classify surfaces. A RANSAC based segmentation algorithm is employed to detect several shapes in a point-cloud. The method is aimed mainly at models containing small numbers of points and shapes as no optimizations or extensions to the general RANSAC framework are adopted.

2.2.4 RANSAC

The RANSAC paradigm extracts shapes by randomly drawing minimal sets from the point data and constructing corresponding shape primitives. A minimal set is the smallest number of points required to uniquely define a given type of geometric primitive. The resulting candidate shapes are tested against all points in the data to determine how many of the points are well approximated by the primitive (called the *score* of the shape). After a given number of trials, the shape which approximates the most points (points well approximated by the primitive are also referred to as *inliers*) is reported, all its inliers are extracted from $\mathcal{P}$ and the algorithm continues on the remaining data. The number of necessary trials is determined such that the probability of a miss is below a given threshold and depends on the number of points belonging to the largest primitive. Since the size of the best primitive is not known a priori, the probability of a miss is computed using the size of the best candidate detected so far as an estimate. RANSAC exhibits the following, desirable properties:

- It is conceptually simple, which makes it easily extensible and straightforward to implement.
- It is very general, allowing its application in a wide range of settings.

- It can robustly deal with data containing more than 50% of outliers [RL93].

Due to the global nature of the algorithm (i.e. both sampling and hypotheses verification operate on the entire point-cloud) its major deficiency is the considerable computational demand if no further optimizations are applied.

Despite these performance issues, several authors have proposed to use this global version of RANSAC for detection of shape primitives. Bolles and Fischler [BF81] apply RANSAC to extract cylinders from range data, Chaperon and Goulette [CG01] use RANSAC and the Gaussian image to find cylinders in 3D point clouds. Both methods, though, do not consider a larger number of different classes of shape primitives. In contrast, Roth and Levine [RL93] describe an algorithm that uses RANSAC to detect a set of different types of simple shapes. However, their method was adjusted to work in the image domain or on range images and they did not provide the optimization necessary for processing large unstructured 3D data sets.

A vast number of extensions to the general RANSAC scheme have been proposed aiming either at improving the robustness or addressing the performance issues. Among the more recent advances, methods such as MLESAC [TZ00] or MSAC [TZ98] improve the robustness of RANSAC with a modified score function, but do not provide any enhancement in the performance of the algorithm, which is the main focus of our work. Nonetheless, it is also possible to use these score functions in our algorithm (see Sec. 2.11).

Performance improvements

The strategies proposed for performance improvements of RANSAC usually either aim at accelerating the process of hypothesis evaluation or employ a modified sampling scheme that increases the probability of creating useful hypotheses such that the required number of overall hypotheses is reduced.

Accelerated evaluation Since RANSAC searches for the shape primitive that achieves maximal score, the general idea for accelerated hypothesis evaluation is to quickly identify those shape primitives that achieve lower score without evaluating all samples in the point-cloud.

A very simple measure to quickly filter out bad hypotheses is the $T_{d,d}$ test proposed by Matas and Chum [MC02]. Each generated primitive is tested on a small random subset of $\mathcal{P}$ comprised of $d \ll |\mathcal{P}|$ points. Only if all d points are well approximated by the primitive the remaining points in $\mathcal{P}$ are also evaluated. They recommend to use $d = 1$ as an optimal setting. Of course, due to the randomized nature of the evaluation, it is possible that also good primitives get rejected by the

$T_{d,d}$ test. Therefore it becomes necessary to generate a larger number of primitives, which however in practice generally turns out to be faster than evaluating all hypotheses on all points.

David Capel [Cap05] proposed to terminate the evaluation of hypotheses as soon as the probability that the currently evaluated hypothesis is better then the so far best found falls below a threshold. Specifically, for a new hypothesis that has been partially evaluated against a subset of n data points, the goal is to determine the probability that having observed $\hat{n}$ inliers so far, the total number of inliers for the new hypothesis is greater than that of the currently best shape primitive. Since the expected number of inliers follows a hypergeometric distribution this is computationally expensive to calculate exactly and instead a lower bound $\hat{n}_{min}$ is derived on the number of inliers observed after evaluating n data points for a given confidence P_{conf}. If, at this point, the number of inliers is below the threshold, bail-out occurs and the hypothesis is discarded. In order to compute the lower bound the hypergeometric distribution is approximated as a binomial distribution for small n or as a normal distribution for larger n. Since new hypotheses are always compared to the currently best model, the order in which hypotheses are generated has a big influence on the effectiveness of the bail-out. The approach proposed in this work on the other hand, while based on similar considerations, always evaluates the most promising hypotheses first which further provides considerable speed-up on average.

A theoretically optimal strategy (if the number of inliers were known a priori) for early termination of hypothesis evaluation was proposed by Chum and Matas [MC05, CM08] based on Wald's theory of sequential decision making [Wal47]. Wald's Sequential Probability Ratio Test (SPRT) is based on the likelihood ratio

$$\lambda_j = \prod_{r=1}^{j} \frac{p(x_r|H_b)}{p(x_r|H_g)} \tag{2.1}$$

where x_r is equal to one if the rth data point is an inlier, and zero otherwise. $p(1|H_g)$ denotes the probability that a randomly chosen data point is an inlier of the best primitive, i.e. it is the fraction κ of inliers contained in $\mathcal{P}$. Similarly, $p(1|H_b)$ is the probability that a randomly chosen data point is an inlier of a bad primitive (i.e. a primitive not achieving maximal score), and this can be modeled using a Bernoulli distribution with parameter δ. If, after evaluating j data points, the likelihood ratio becomes greater than some threshold A, the model is rejected. The decision threshold A can be set to achieve optimal running time given κ and δ [CM08]. Since the two parameters κ and δ have to be estimated during the evaluation process, optimality is no longer guaranteed and in practice SPRT and Capel's bail-out achieve comparable performance [RFP08]. Note that similarly to the bail-out test, SPRT also depends on the order in which hypotheses are considered because the estimates for κ and δ are adjusted after rejection or acceptance of a hypothesis respectively.

Improved sampling If there is a priori information on the reliability of the points in $\mathcal{P}$ it may make sense to exploit this knowledge in order to generate better hypotheses.

The Progressive Sample Consensus (PROSAC) suggested by Chum and Matas [CM05] creates the same hypotheses as unmodified RANSAC but uses the additional information such that hypotheses are estimated first from data points with high accuracy. Therefore there is in general an increased likelihood of generating the best model early on in the process. This reduces the required number of overall hypotheses and also increases the effectiveness of both the bail-out test or SPRT. Our method however only has a weak dependency on the order in which hypotheses are generated and the effect of PROSAC can therefore be expected to be relatively small.

A similar, yet somewhat simpler approach was suggested by Tordoff and Murray [TM02] who use the additional information on the points' quality to design a guided sampling. The sampling prefers points of high quality which therefore are selected into minimal sets for estimation of new hypothesis more frequently. They showed empirically that the runtime was drastically reduced because good hypotheses were created much earlier.

A modified sampling approach that does not require any a priori information is the locally optimized RANSAC proposed by Chum et al. [CMK03]. Usually RANSAC operates under the assumption that a hypothesis computed from any samples that are inliers of the best possible model results in exactly this best model. However, for noisy data that is often not the case as the estimated model is distorted by the noise. The locally optimized RANSAC is designed to address this issue. Since, despite the noise, a model estimated from a sample of inliers does indeed tend to be consistent with a significant fraction of all the inliers, a small constant number of hypotheses are generated using only the set of inliers to the current best model. Note that these hypotheses do not necessarily have to be generated from minimal subsets. In addition to providing a more robust fit, this sampling technique has the effect of improving the consensus score more rapidly than standard RANSAC and therefore results in faster termination.

Real-time approaches In a real-time setting it is mandatory to bound the runtime of the algorithm. To this end, David Nistér [Nis05] proposes to fix the number of generated hypotheses in advance. All hypotheses are scored on a subset of the data in parallel. The hypotheses are ordered according to the achieved score and only a given fraction is retained and evaluated on the next subset. This is repeated until only a single hypothesis remains or all subsets have been considered.

This approach was subsequently refined by Raguram et al. [RFP08] in order to allow the generation of fewer hypotheses in case there is a high inlier ratio. The maximal number of allowed hypotheses remains bounded however.

In our case wee seek an unknown large number of possibly very small shapes in huge point-clouds and the amount of necessary candidate primitives cannot be reasonably specified in advance.

2.3 Overview

Given a point-cloud $\mathcal{P} = \{p_1, \ldots, p_N\}$ with associated normals $\{n_1, \ldots, n_N\}$ the output of our algorithm is a set of primitive shapes $\Psi = \{\psi_1, \ldots, \psi_n\}$ with corresponding disjoint sets of points $\mathcal{P}_\Psi = \{\mathcal{P}_{\psi_1} \subset \mathcal{P}, \ldots, \mathcal{P}_{\psi_n} \subset \mathcal{P}\}$ and a set of remaining points $\mathcal{R} = \mathcal{P} \backslash \bigcup_\psi \mathcal{P}_\psi$. Similar to Roth and Levine [RL93] and Décoret et al. [DDSD03], we frame the shape extraction problem as an optimization problem defined by a score function. The overall structure of our method is outlined in pseudo-code in Algorithm 1. In each iteration of the algorithm, the primitive with maximal score is searched using the RANSAC paradigm. New shape candidates are generated by randomly sampling minimal subsets of $\mathcal{P}$ using our novel sampling strategy (see Sec. 2.6). Candidates of *all* considered shape types are generated for *every* minimal set and all candidates are collected in the set $\mathcal{C}$. Thus no special ordering has to be imposed on the detection of different types of shapes. After new candidates have been generated, the candidate m with the highest score is computed employing the efficient lazy score evaluation scheme presented in Sec. 2.8. The best candidate is only accepted if, given the number of inliers $|m|$ of the candidate and the number of drawn candidates $|\mathcal{C}|$, the probability $P(|m|, |\mathcal{C}|)$ that no better candidate was overlooked during sampling is high enough (see Sec. 2.5.1). We provide an analysis of our sampling strategy to derive a suitable probability computation. If a candidate is accepted, the corresponding points $\mathcal{P}_m$ are removed from $\mathcal{P}$ and the candidates $\mathcal{C}_m$ generated with points in $\mathcal{P}_m$ are deleted from $\mathcal{C}$. The algorithm terminates as soon as $P(\tau, |\mathcal{C}|)$ for a user defined minimal shape size τ is large enough.

The following discussion is based on a standard score function that counts the number of compatible points for a shape candidate [RL93, GBS03], we will later show how alternative score functions can also be integrated in our setting. The standard score function has two free parameters: ϵ specifies the maximum distance of a compatible point while α restricts the deviation of a points' normal from that of the shape. We also ensure that only points forming a connected component on the surface are considered (see Sec. 2.7).

Algorithm 1 Extract shapes in the point cloud $\mathcal{P}$

```
$\Psi \leftarrow \emptyset$ {extracted shapes}
$\mathcal{C} \leftarrow \emptyset$ {shape candidates}
repeat
  $\mathcal{C} \leftarrow \mathcal{C} \cup$ newCandidates() {see Sec. 2.4 and 2.6}
  $m \leftarrow$ bestCandidate($\mathcal{C}$) {see Sec. 2.7}
  if $P(|m|, |\mathcal{C}|) > p_t$ then
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \mathcal{P}_m$ {remove points}
    $\Psi \leftarrow \Psi \cup m$
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{C}_m$ {remove invalid candidates}
  end if
until $P(\tau, |\mathcal{C}|) > p_t$
return $\Psi$
```

2.4 Shape estimation

As mentioned above, the shapes we consider in this work are planes, spheres, cylinders, cones and tori which have between three and seven parameters. Every 3D-point p_i fixes only one parameter of the shape. In order to reduce the number of points in a minimal set we require an unoriented approximate surface normal n_i for each point (see Sec. 1.5.1), so that the direction gives us two more parameters per sample. That way it is possible to estimate each of the considered basic shapes from at most three point samples. However, always using one additional sample is advantageous because the surplus parameters can be used to immediately verify a candidate and thus eliminate the need of evaluating many relatively low scored shapes [MC02].

Plane For a plane, $\{p_1, p_2, p_3\}$ constitutes a minimal set when not taking into account the normals in the points. To confirm the plausibility of the generated plane, the deviation of the plane's normal from n_1, n_2, n_3 is determined and the candidate plane is accepted only if all deviations are less than the predefined angle α.

Sphere A sphere is fully defined by two points with corresponding normal vectors. We use the midpoint of the shortest line segment between the two lines given by the points p_1 and p_2 and their normals n_1 and n_2 to define the center of the sphere c. We take $r = \frac{\|p_1 - c\| + \|p_2 - c\|}{2}$ as the sphere radius. The sphere is accepted as a shape candidate only if all three points are within a distance of ϵ of the sphere and their normals do not deviate by more than α degrees.

Cylinder To generate a cylinder from two points with normals we first establish the direction of the axis with $a = n_1 \times n_2$. Then we project the two parametric lines $p_1 + tn_1$ and $p_2 + tn_2$ along the axis onto the $a \cdot x = 0$ plane and take their intersection as the center c. We set the radius to the distance between c and p_1 in that plane. Again the cylinder is verified by applying the thresholds ϵ and α to distance and normal deviation of the samples.

Cone Although the cone, too, is fully defined by two points with corresponding normals, for simplicity we use all three points and normals in its generation. To derive the position of the apex c, we intersect the three planes defined by the point and normal pairs. Then the normal of the plane defined by the three points $\{c + \frac{p_1 - c}{\|p_1 - c\|}, \dots, c + \frac{p_3 - c}{\|p_3 - c\|}\}$ gives the direction of the axis a. Now the opening angle ω is given as $\omega = \frac{\sum_i \arccos((p_i - c) \cdot a)}{3}$. Afterwards, similar to above, the cone is verified before becoming a candidate shape.

Torus Just as in the case of the cone we use one more point than theoretically necessary to ease the computations required for estimation, i.e. four point and normal pairs. The rotational axis of the torus is found as one of the up to two lines intersecting the four point-normal lines $p_i + \lambda n_i$ [MLM01]. To choose between the two possible axes, a full torus is estimated for both choices and the one which causes the smaller error in respect to the four points is selected. To find the minor radius, the points are collected in a plane that is rotated around the axis. Then a circle is computed using three points in this plane. The major radius is given as the distance of the circle center to the axis.

2.5 Complexity

The complexity of RANSAC is dominated by two major factors: The number of minimal sets that are drawn and the cost of evaluating the score for every candidate shape. As we desire to extract the shape that achieves the highest possible score, the number of candidates that have to be considered is governed by the probability that the best possible shape is indeed detected, i.e. that a minimal set is drawn that defines this shape.

2.5.1 Probabilities

Consider a point cloud $\mathcal{P}$ of size N and a shape ψ therein consisting of n points. Let k denote the size of a minimal set required to define a shape candidate. If we assume that any k points of the shape will lead to an appropriate candidate shape then the probability of detecting ψ in a single pass is:

$$P(n) = \binom{n}{k} \Big/ \binom{N}{k} \approx \left(\frac{n}{N}\right)^k \tag{2.2}$$

The probability of a successful detection $P(n, s)$ after s candidates have been drawn equals the complementary of s consecutive failures:

$$P(n, s) = 1 - (1 - P(n))^s \tag{2.3}$$

Solving for s tells us the number of candidates T required to detect shapes of size n with a probability $P(n, T) \geq p_t$:

$$T \geq \frac{\ln(1 - p_t)}{\ln(1 - P(n))} \tag{2.4}$$

For small $P(n)$ the logarithm in the denominator can be approximated by its Taylor series $\ln(1 - P(n)) = -P(n) + O(P(n)^2)$ so that:

$$T \approx \frac{-\ln(1 - p_t)}{P(n)} \tag{2.5}$$

Given the cost C of evaluating the score function, the asymptotic complexity of the RANSAC approach is

$$O(TC) = O(\frac{1}{P(n)}C) \tag{2.6}$$

.

2.6 Sampling strategy

As can be seen from the last formula, the runtime complexity is directly linked to the success rate of finding good sample sets. Therefore we will now discuss in detail how sampling is performed.

2.6.1 Localized sampling

Since shapes are local phenomena, the a priori probability that two points belong to the same shape is higher the smaller the distance between the points. In our sampling strategy we want to exploit this fact to increase the probability of drawing minimal sets that belong to the same shape. In a different context, Myatt et al. [MTN+02] have shown that non-uniform sampling based on locality leads to a significantly increased probability of selecting a set of inliers. From a ball of given radius around an initially unrestrainedly drawn sample the remaining samples are picked to obtain a complete minimal set. This requires to fix a radius in advance, which they derive from a known (or assumed) outlier density and distribution. In our setup however, outlier density and distribution vary strongly for different

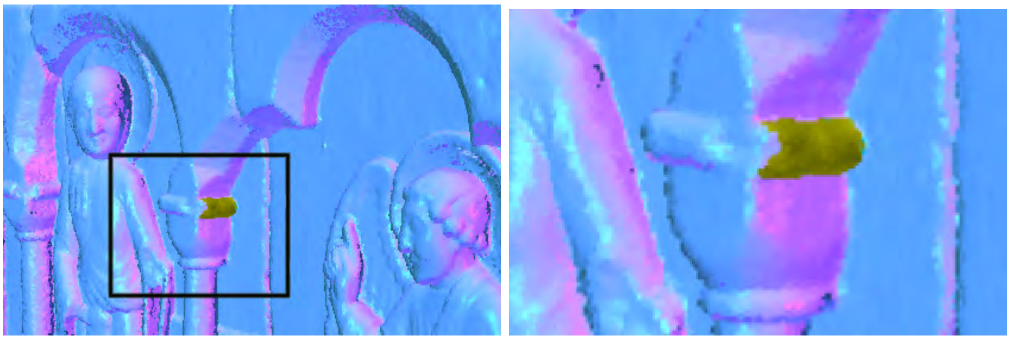


Figure 2.1: A small cylinder that has been detected by our method. The shape consists of 1066 points and was detected among 341,587 points. That corresponds to a relative size of $1/3000$.

models and even within in a single model, which renders a fixed radius inadequate. Also, in our case, using minimal sets with small diameter introduces unnecessary stability issues in the shape estimation procedure for shapes that could have been estimated from samples spread farther apart. Therefore, we propose a novel sampling strategy that is able to adapt the diameter of the minimal sets to both, outlier density and shape size.

We use an octree to establish spatial proximity between samples very efficiently. When choosing points for a new candidate, we draw the first sample p_1 without restrictions among all points. Then a cell C is randomly chosen from any level of the octree such that p_1 is contained in C. The $k-1$ other samples are then drawn only from within cell C.

The effect of this sampling strategy can be expressed in a new probability $P_{local}(n)$ for finding a shape ψ of size n:

$$P_{local}(n) = P(p_1 \in \psi)P(p_2 \dots p_k \in \psi | p_2 \dots p_k \in C) \tag{2.7}$$

The first factor evaluates to n/N. The second factor obviously depends on the choice of C. C is well chosen if it contains mostly points belonging to ψ. The existence of such a cell is backed by the observation that for most points on a shape, except on edges and corners, there exists a neighborhood such that all of the points therein belong to that shape. Although in general it is not guaranteed that this neighborhood is captured in the cells of the octree, in the case of real-life data, shapes have to be sampled with an adequate density for reliable representation and, as a consequence, for all but very few points such a neighborhood will be at least as large as the smallest cells of the octree. For the sake of analysis, we assume that there exists a C for every $p_i \in \psi$ such that ψ will be supported by half of the points in C, which accounts for up to 50% local noise and outliers. We conservatively estimate the probability of finding a good C by $\frac{1}{d}$ where d is the depth of the octree (in practice a subtree of cells rooted in the highest good cell will be good as well). The conditional probability for $p_2, p_3 \in \psi$ in the case of a good cell is then described by $\frac{\binom{|C|/2}{k-1}}{\binom{|C|}{k-1}} \approx (\frac{1}{2})^{k-1}$. And substituting yields:

$$P_{local}(n) = \frac{n}{Nd2^{k-1}} \tag{2.8}$$

As large shapes can still be estimated using points spread far apart in large octree cells, the stability of the shape estimation does not suffer from the localized sampling strategy.

The impact of this sampling strategy is best illustrated with an example. The cylinder depicted in Figure 2.1 consists of 1066 points. At the time that it belongs to one of the largest shapes in the point-cloud, 341,547 points of the original 2 million still remain. Thus, it then comprises only three thousandth of the point-cloud. If an ordinary uniform sampling strategy were to be applied, 151,522,829 candidates would have to be drawn to achieve a detection probability of 99%. With our strategy only 64,929 candidates have to be generated for the same probability. That is an improvement by three orders of magnitude, i.e. in this case that is the difference between hours and seconds.

Level weighting Choosing C from a proper level is an important aspect of our sampling scheme. Here we propose a simple heuristic to further improve the sampling efficiency: We choose C from a level according to a non-uniform distribution that reflects the likelihood of the respective level to contain a good cell. To this end, the probability P_l of choosing C from level l is first initialized with $\frac{1}{d}$. Then for every level l, we keep track of the sum σ_l of the scores achieved by the candidates generated from a cell on level l. After a given number of candidates has been tested, a new distribution for the levels is computed. The new probability $\hat{P}_l$ of the level l is given as

$$\hat{P}_l = x\frac{\sigma_l}{wP_l} + (1-x)\frac{1}{d}\,, \tag{2.9}$$

where $w = \sum_{i=1}^{d} \frac{\sigma}{P_i}$. We set $x = .9$ to ensure that at all times at least 10% of the samples are spread uniformly over the levels to be able to detect when new levels start to become of greater importance as more and more points are removed from $\mathcal{P}$.

2.6.2 Number of candidates

In Section 2.5 we gave a formula for the number of candidates necessary to detect a shape of size n with a given probability. However, in our case, the size n of the largest shape is not known in advance. Moreover, if the largest candidate has been generated early in the process we should be able to detect this lucky case and extract the shape well before achieving a precomputed number of candidates while on the other hand we should use additional candidates if it is still unsure that indeed the best candidate has been detected. Therefore, instead of fixing the number of candidates, we repeatedly analyze small numbers t of additional candidates and consider the best one ψ_m generated so far each time. As we want to achieve a low probability that a shape is extracted which is not the real maximum, we observe the probability $P(|\psi_m|, s)$ with which we would have found another shape of the same size as ψ_m. Once this probability is higher than a threshold p_t (we use 99%) we conclude that there is a low chance that we have overlooked a better candidate and extract ψ_m. The algorithm terminates as soon as $P(\tau, s) > p_t$.

2.7 Score

The score function $\sigma_{\mathcal{P}}$ is responsible for measuring the quality of a given shape candidate. We use the following aspects in our scoring function:

- To measure the support of a candidate, we use the number of points that fall within an ϵ-band around the shape.
- To ensure that the points inside the band roughly follow the curvature pattern of the given primitive, we only count those points inside the band whose normals do not deviate from the normal of the shape more than a given angle α.
- Additionally we incorporate a connectivity measure: Among the points that fulfill the previous two conditions, only those are considered that constitute the largest connected component on the shape.

More formally, given a shape ψ whose fidelity is to be evaluated, $\sigma_{\mathcal{P}}$ is defined as follows:

$$\sigma_{\mathcal{P}}(\psi) = |\mathcal{P}_\psi| \,, \tag{2.10}$$

i.e. we count the number of points in $\mathcal{P}_\psi$. $\mathcal{P}_\psi$ is defined in the following two steps:

$$\hat{\mathcal{P}_\psi} = \{p | p \in \mathcal{P} \wedge |d(\psi, p)| < \epsilon \wedge \arccos(|n(p) \cdot n(\psi, p)|) < \alpha\} \tag{2.11}$$

$$\mathcal{P}_\psi = maxcomponent(\psi, \hat{\mathcal{P}_\psi}) \,, \tag{2.12}$$

where $d(\psi, p)$ is the signed distance of point p to the shape primitive ψ (see Sec. 1.5.3), $n(p)$ is the normal in p and $n(\psi, p)$ is the normal of ψ in p's projection on ψ. $maxcomponent(\psi, \hat{\mathcal{P}_\psi})$ extracts the group of points in $\hat{\mathcal{P}_\psi}$ whose projections onto ψ belong to the largest connected component on ψ.

2.7.1 Connected components

We find connected components in a bitmap located in the parameter domain of the shape. A pixel in the bitmap is set if a point is projected into it. Ideally, the size β of the pixels in the bitmap should correspond to the distance between neighboring points in the data, i.e. the sampling resolution. If the data is irregularly sampled, β should be chosen as the minimal sampling resolution satisfied everywhere in the data.

We use the parameterizations given in Sec. 1.5.3 for the bitmaps. Connected components are computed with accordant wrapping in order to deal with the discontinuities in the parameterizations of spheres, cylinders, cones and tori. In some cases, e.g. for cylinders with very large radii, it may be that, due to the discontinuity, the bitmap contains very large empty parts. Instead of allocating large bitmaps in such cases, we resort to a hashtable representation of the bitmap which contains only the set pixels. This turned out to be faster and more flexible then to adjust the parametrization.

2.8 Score evaluation

The second major performance factor of the RANSAC scheme is the score function evaluation. In our case, in a naïve implementation, the distance to all points in $\mathcal{P}$ would have to be computed together with a normal at a corresponding position on the shape for each candidate. Then the largest connected component has to be found among all compatible points.

2.8.1 Random subsets

Obviously the cost of evaluation would be prohibitive without any optimizations. But since in each run we are only interested in the candidate that achieves the highest score, using the entire point cloud $\mathcal{P}$ when computing $\sigma_{\mathcal{P}}(\psi)$ is not necessary for every shape candidate. We significantly reduce the number of points that have to be considered in the evaluation of $\sigma_{\mathcal{P}}(\psi)$ by splitting the point cloud $\mathcal{P}$ into a set of disjoint random subsets: $\mathcal{P} = \mathcal{S}_1 \bigcup \ldots \bigcup \mathcal{S}_r$.

After a shape candidate was generated and successfully verified, the candidate is only scored against the first subset $\mathcal{S}_1$ and no connected component is extracted yet. From the score $\sigma_{\mathcal{S}}(\psi)$ on a subset $\mathcal{S} \subset \mathcal{P}$ an estimate $\hat{\sigma}_{\mathcal{P}}(\psi)$ for the score $\sigma_{\mathcal{P}}(\psi)$ on all points can be extrapolated using the well known induction from inferential statistics:

$$\hat{\sigma}_{\mathcal{P}}(\psi, \mathcal{S}) = -1 - f(-2 - |\mathcal{S}|, -2 - |\mathcal{P}|, -1 - |\mathcal{S}_\psi|) \, , \tag{2.13}$$

where

$$f(N, x, n) = \frac{xn \pm \sqrt{\frac{xn(N-x)(N-n)}{N-1}}}{N} \tag{2.14}$$

is the mean plus/minus the standard deviation of the hypergeometric distribution. $\hat{\sigma}_{\mathcal{P}}(\psi)$ is a confidence interval $[l_\psi, u_\psi]$ that describes a range of likely values for the true score $\sigma_{\mathcal{P}}(\psi)$. The expected value $E(\sigma_{\mathcal{P}}(\psi))$ is given by $\frac{l_\psi + u_\psi}{2}$. With this extrapolation the potentially best candidate ψ_m can be quickly identified by choosing the one with the highest expected value. Since the uncertainty of the estimation is captured in the confidence intervals, the truly maximal candidate can be found by comparing the confidence intervals of the candidates.

If the confidence intervals of ψ_m and another candidate ψ_i overlap, the score on an additional subset is evaluated for both candidates and new extrapolations are computed, now taking into account the scores on all subsets that have already been computed:

$$\hat{\sigma}_{\mathcal{P}}(\psi) = \hat{\sigma}(\psi, \bigcup_i \mathcal{S}_i) \tag{2.15}$$

The more subsets have been considered, the smaller becomes the range of the confidence intervals, since the uncertainty in the estimation decreases. Further subsets are included until the confidence intervals of ψ_i and ψ_m no longer overlap and it can be decided if either ψ_i or ψ_m is better.

To include the effect of the connectedness condition in the extrapolation, every time an additional subset has been evaluated, the maximal connected component is found among all the compatible points that have been discovered so far. The resolution of the bitmap that is used to find the components has to be adapted to reflect the lower sampling rate of the subsets. If $\frac{1}{x} = \frac{\sum_i |\mathcal{S}_i|}{|\mathcal{P}|}$ is the fraction of points in $\mathcal{P}$ that have been tested so far, then the bitmap resolution is adjusted to $x\beta$.

Pseudocode for the score evaluation is given in Alg. 2. In line 1 a heap $\mathcal{H}$ containing all candidates is constructed such that candidates with higher upper bound u_ψ will be popped first. In lines 3 - 19 processing of candidates continues until the heap is empty or an eligible best candidate was found. A candidate m is eligible if its score is greater than the minimum required score τ and the probability $P(u_m, s)$ (where s is the number of candidates drawn so far) is greater than p_t. In lines 5-7 the score estimate of the currently best candidate m is refined as long as additional subsets $\mathcal{S} \subset \mathcal{P}$ exist and the lower bound l_m is larger than the upper bound u_f of the next best candidate f. If after the refinement the candidate m turns out to be no longer eligible, the next candidate f is considered (lines 8-9). If the next candidate is also not eligible, no eligible candidate remains and the algorithm returns signalling failure (lines 9-13). If on the other hand the lower bound l_m of the candidate m after refinement is still larger than the upper bound

Algorithm 2 Find best candidate in candidate set $\mathcal{C}$

```
1:  $\mathcal{H} \leftarrow$ makeHeap($\mathcal{C}$)
2:  $m \leftarrow$ popHeap($\mathcal{H}$)
3:  while $|\mathcal{H}| > 0$ do
4:      $f \leftarrow$ popHeap($\mathcal{H}$)
5:      while $l_m \neq u_m \wedge u_f \leq l_m$ do
6:          improveBounds($m$)
7:      end while
8:      if $u_m < \tau \vee P(u_m, s) > p_t$ then
9:          $m \leftarrow f$
10:         if $u_m < \tau \vee P(u_m, s) > p_t$ then
11:             return $\emptyset$
12:         end if
13:     end if
14:     if $l_m > b_f$ then
15:         return $m$
16:     end if
17:     $\mathcal{H} \leftarrow$ pushHeap($m, \mathcal{H}$)
18:     $m \leftarrow f$
19: end while
```

u_f of the next best candidate, then m is the best candidate and is returned (lines 14-15). Otherwise m is pushed back onto the heap and the same procedure is repeated for the next best candidate (lines 17-18).

The advantage of this priority based candidate evaluation is that it is less dependent on the random order in which candidates are generated. Compared to the order dependent approach of David Capel [Cap05] (see Sec. 2.2.4) we achieve a 20% speedup on average on a single core machine. Yet, admittedly our approach does not scale well with the number of cores. Indeed, already on a dual core machine the concurrent, yet order dependent evaluation is on par with a parallelized version of our algorithm. This is due to the maintenance of the heap structure which is not well suited for parallelization.

2.8.2 Octree

In order to accelerate the collection of the compatible points for a primitive ψ, for each of the subsets $\mathcal{S}_i \subset \mathcal{P}$ an octree is constructed, so that during the cost function evaluation only the points lying in cells within ϵ distance to the shape have to be considered. To this end, the octree is traversed recursively and a cell is only visited if the distance of ψ to the cell's center is less than half the cell's diameter plus ϵ.

2.9 Refitting

When a candidate shape ψ has been selected for extraction, a refitting step is executed before the shape is finally accepted. As is the standard in RANSAC based algorithms, we use a least-squares approach [Sha98]. This optimizes the geometric error of the candidate shape. In refitting and extraction we include all compatible points within a distance of 3ϵ from the shape, as this removes unnecessary clutter from the point-cloud [GBS03].

2.10 Out-of-core detection

Since the run-time complexity of the detection algorithm is inversely proportional to the probability $P(n)$ for detecting a shape of size n (see Eq. (2.6)), this may, even despite our local sampling strategy, become a problem on very large point-clouds such as e.g. huge city models. For instance, in the case of scanned city we have to deal with thousands of individual buildings, most of which are of similar size. Let us consider for example an aerial scan with a spatial resolution of $7cm$ of the city of Munich roughly containing $200,000$ individual buildings and roughly being spread over about $300km^2$. In this case the total raw data size consists of $61 \cdot 10^9$ points. For simplicity let us further assume that each building consists of only 6 individual faces with a total surface area of approximately $1000m^2$ resulting in about $32,000$ points per primitive shape. To find one of the individual faces with a probability of 99% with the localized RANSAC approach as described above, we would have to draw about 38 million different minimal sets from the point-cloud which is still too much, especially since for each of the resulting individual shapes the score function must be evaluated. One way to solve this problem is to adapt the size of the primitive to be extracted to the total number of points in the model, i.e. to keep the ration between the primitive size n and the number N of points in the point-cloud bounded, which is discussed in detail in the following section.

2.10.1 Maximal primitive extent

Since buildings typically do not exceed a maximal side length, we can safely apply our original RANSAC scheme to local parts of the input point-cloud. Such local parts have to be large enough so that no structures in the data can be missed. Moreover, to be effective, these parts should be chosen such that the ratio $1/P(n)$

is bounded locally. Since arbitrarily small shapes can appear in general, the local parts have to be hierarchical. To construct such local parts of the input point-cloud, we propose to sort the point-cloud into a global, out-of-core octree data structure, but other hierarchies might be suitable as well.

In principle, the shape detection is then executed for each of the cells in the octree. Of course, some additional care has to be taken to ensure sufficient overlap between cells. Also the octree should be constructed in a way such that the side lengths of the cells correspond to the expected maximal side length of structures in the point data.

The octree data structure

The side length of the largest cells is computed based on a maximal area A and a maximal aspect ratio R of the surfaces contained in the data. These parameters need to be specified by the user. The side length S of the cells is then given as $S = \sqrt{R \cdot A}$. The bounding cube of the octree is chosen such that on some level L the side length of the contained cells is exactly S. In order to achieve overlap between cells, when the shape detection is executed for cell C, the points in neighboring cells on level $L + 1$ are included during the shape detection as well (these have side length $1/2S$). The minimal size of a shape is set to $1/4A$ during the detection, as smaller shapes will be searched on the next finer level in the octree. This way the ratio $1/P(n)$ is effectively bounded (since the side length of a cell and the minimal size of a shape are coupled via the hierarchy) and the data can be processed out-of-core due to the local nature of the algorithm. Thus, once the detection has been executed on level L, we continue in the same manner on level $L + 1$ until all shapes have been detected.

2.11 Alternate score

The discussion so far has only considered the standard score function using the criteria described in Sec. 2.7. While this scoring method already reliably detects most primitives (see Sec. 2.12) there remain two issues: Firstly, this standard scoring function does not differentiate between different types of primitives, i.e. if a subset of $\mathcal{P}$ is representable by two different types of primitives the detected type of primitive can be either one. Thus, if there are several similar such subsets in $\mathcal{P}$, the separate, yet similar, subsets may be inconsistently represented by primitives of different type. Secondly, the standard scoring function successfully ignores outliers but does not specifically consider noise present in the data, i.e. the standard scoring function treats all points within the distance threshold ϵ equally, regardless of the actual distance from the primitive. Therefore the scoring function does only weakly distinguish between primitives based on their approximation quality - only a fixed threshold is considered, more subtle variations and noise are ignored.

In [TZ98] Torr and Zisserman, following a similar line of argumentation, proposed MSAC in order to alleviate the influence of a fixed threshold. The scoring method presented in this section is an extension of their idea and, besides addressing noise, also differentiates between primitive types. Both these aspects can be effectively handled using a model selection criterion. Model selection is a branch of statistics and information theory [BA02] which is concerned with finding the right parametric model for a given set of data. A scoring function is used to evaluate different models that were fitted to the data. The model with the best score is then selected as the most appropriate one. The scoring criterion aims to find a trade-off between the complexity of the representation, i.e. the number of parameters used in the model, and the residual error, i.e. the approximation quality. In the model selection literature there is a large body of different criterions to choose from.

One of the popular scoring criteria is Akaike's *An Information Criterion* (AIC) [Aka73] which minimizes the expected entropy of yet unobserved data, but suffers from a tendency to overfit [LB87]. Other popular model selection criteria are based on minimizing the coding length of the observed data with respect to the employed models: Wallace's *Minimum Message Length* (MML) [WB68] and Rissanen's very similar, but independently developed, *Minimum Description Length* (MDL) [Ris78]. Although based on an Bayesian standpoint the *Bayesian Information Criterion* (BIC) by Schwarz [Sch78] also leads to a very similar measure. However, in practice most of these models and their variants behave very similarly and the differences are mostly negligible [BHG06]. Given the small practical differences, we base the remainder of this discussion on the MDL criterion as it provides a small set of relatively simple and intuitive parameters for controlling the trade-off between complexity and error. However, other criteria would be applicable as well.

2.11.1 Minimum Description Length

The MDL criterion is based on the following cost function $C(\psi, \mathcal{P}_\psi)$:

$$C(\psi, \mathcal{P}_\psi) = L(\mathcal{P}_\psi|\psi) + L(\psi) \tag{2.16}$$

where $\mathcal{P}_\psi$ is defined as in Eq. (2.12), $L(\mathcal{P}_\psi|\psi)$ denotes the coding length required for describing the deviation of $\mathcal{P}_\psi$ from the given primitive ψ (usually called residuals) and $L(\psi)$ is the coding length for the primitive parameters respectively. Commonly $L(\mathcal{P}_\psi|\psi)$ is defined as $L(\mathcal{P}_\psi|\psi) = \frac{1}{2\sigma^2}RSS$ in the MDL framework, where RSS denotes the residual sum of squares and σ^2 the noise variance in the data [HY01]. Thus in our setting, one would ideally like to base $L(\mathcal{P}_\psi|\psi)$ on the distance between ψ and $\mathcal{S}(\mathcal{P}_\psi)$, i.e.

$$L(\mathcal{P}_\psi|\psi) = \alpha \int_{s \in \mathcal{S}(\mathcal{P}_\psi)} d(s,\psi)^2 ds \tag{2.17}$$

where $\mathcal{S}(\mathcal{P}_\psi)$ denotes a continuous surface represented by the point-cloud $\mathcal{P}_\psi$ (e.g. the MLS surface introduced in Sec. 1.5.2), α is an application dependent weighting factor and $d(s,\psi)$ denotes the distance of s to ψ (see Sec. 1.5.3). Since computing $\int_{s \in \mathcal{P}_\psi} d(s,\psi)^2 ds$ exactly is very involved we resort to a simple approximation (this is a common approximation, similar for instance to [HDD+93]):

$$L(\mathcal{P}_\psi|\psi) = \alpha \sum_{p_j \in \mathcal{P}_\psi} d(p_j,\psi)^2. \tag{2.18}$$

The MDL criterion also assigns costs to unassigned points, i.e. all points collected in the set of remaining points $\mathcal{R}$. The cost for these points $L(\mathcal{R}|\emptyset)$ is defined as

$$L(\mathcal{R}|\emptyset) = \gamma|\mathcal{R}| \tag{2.19}$$

where γ again is a user specified factor.

Finally, $L(\psi)$ considers the complexity of ψ and is defined as

$$L(P_i) = \beta|\psi| \tag{2.20}$$

where $|\psi|$ denotes the number parameters of ψ and β again is an application dependent weighting factor. We set $L(\emptyset)$ to zero.

Given the cost function $C(\psi, \mathcal{P}_\psi)$ for a single primitive we can define the cost $C(\Psi, \mathcal{P}_\Psi)$ of the entire partitioning found by the primitive detection as follows:

$$C(\Psi, \mathcal{P}_\Psi) = \sum_{\psi \in \Psi} C(\psi, \mathcal{P}_\psi) + C(\emptyset, \mathcal{R}) \tag{2.21}$$

The goal of the primitive detection is now to minimize the energy of Eq. (2.21). To this end, in principle we have to simultaneously determine the number of partitions $|\Psi|$, the shape of the partitions $\mathcal{P}_\Psi = \{\mathcal{P}_{\psi_i}\}$ and the optimal type and parameters of the primitives ψ_i - which all evidently are very tightly coupled. It is quite clear that even for a relatively simple input point-cloud $\mathcal{P}$, it is computationally not feasible to explore all possible partitions and test all possible primitives in order to find the optimal solution.

Here we follow a simple greedy strategy [LGB95] which nonetheless proves to be effective in practice. In fact, we can greedily minimize Eq. (2.21) using the RANSAC algorithm introduced in the previous section using a suitable score function which is presented in the following. Recently, we also proposed an approach to jointly optimize the number of primitives, the partitioning and the primitives' type and parameters [LSSK09]. While the joint optimization indeed outperforms the simple greedy RANSAC approach and results in an improved partitioning - especially on generic surfaces - it unfortunately has very long runtimes and is therefore not suitable for larger point-clouds. The strategy employed here on the other hand directly benefits from the efficiency of our RANSAC algorithm and is applicable even to very large point-clouds.

Since RANSAC tries to maximize the score, we cannot directly employ $C(\psi, \mathcal{P}_\psi)$ as the score of a primitive ψ. However, it is straightforward to derive a score based on the *benefit* of a given primitive:

$$\begin{aligned}
\sigma'_{\mathcal{P}}(\psi) &= C(\emptyset, \mathcal{P}_\psi) - C(\psi, \mathcal{P}_\psi) && (2.22) \\
&= L(\mathcal{P}_\psi|\emptyset) + L(\emptyset) - L(\mathcal{P}_\psi|\psi) - L(\psi) && (2.23) \\
&= \gamma|\mathcal{P}_\psi| - \alpha \sum_{p_j \in \mathcal{P}_\psi} d(p_j, \psi)^2 - \beta|\psi| && (2.24)
\end{aligned}$$

Parameters

The choice of the parameters in this MDL score function can be based on the following intuition: If γ is set to one, then β can be seen as controlling the number of points a proxy has to subsume in order to become beneficial. For instance, a sphere has four parameters and therefore has to approximate at least 4β points in order to be worthwhile (The choice of β in general is quite subjective and in our experiments we always use $\beta = 10$ as we found this to give a reasonable tradeoff). However, the degree to which each point is credited to the proxy depends on the approximation error and is controlled by the parameter α. Intuitively α controls the desired error of the resulting approximation. In practice, if we desire an average approximation error of κ we set $\alpha = \frac{1}{(3\kappa)^2}$. Given $\gamma = 1$ this means that any point farther than 3κ from a primitive will not be beneficial anymore. Therefore we can use this distance as the threshold $\epsilon = 3\kappa$ during the score evaluation.

Extrapolation

To use the MDL score function in our RANSAC approach it is necessary to extrapolate the score value computed on a subset to the score on the entire point-cloud. For the standard score function it was sufficient to extrapolate the size using Eq. (2.13), i.e. the number of points in the support region, of a candidate primitive in order to extrapolate the score. We propose the following extrapolation for the

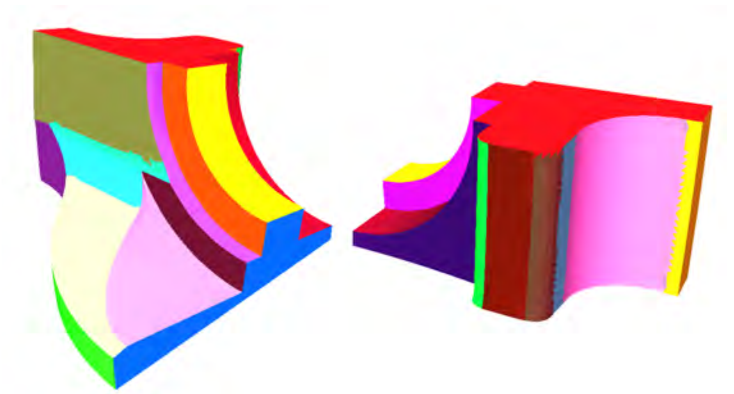


Figure 2.2: To generate this image our algorithm was applied to the barycenters of the triangles. The triangles were than colored according to the shape of their barycenter and the vertices were projected onto the shape. The jagged lines appear because the triangulation does not contain the edges of the shapes.

score defined by Eq. (2.22):

$$\hat{\sigma}'_{\mathcal{P}}(\mathcal{S},\psi) = \frac{L(\mathcal{S}_\psi|\emptyset) - L(\mathcal{S}_\psi|\psi)}{|\mathcal{S}_\psi|}\hat{\sigma}_{\mathcal{P}}(\mathcal{S},\psi) - L(\psi), \tag{2.25}$$

where $\hat{\sigma}_{\mathcal{P}}(\mathcal{S},\psi)$ is the extrapolated size defined in Eq. (2.13).

2.12 Results

We have run extensive tests of our algorithm on different kinds of geometry. The results show that basic shapes are reliably detected if they are present in the data. For parts of a surface that closely resemble a basic shape, a well approximating representation is obtained. More involved areas are partitioned into basic shapes in a reasonable manner and the number of remaining points reflects the complexity of the surface. The algorithm exhibits high performance as is shown in the timings of Table 2.1.

To illustrate the effect of the two major optimizations employed by our algorithm, we disabled them independently and compare the resulting timings to the optimized version in Fig. 2.3. The timings were obtained with the parameters given in Table 2.1. The gain of the localized sampling strategy depends on the relative sizes of the shapes with respect to the model. The localized sampling improves the performance especially for smaller shapes detected later on in the process, as their relative size decreases and a global sampling strategy requires far more candidates. Conversely, the score evaluation on subsets has a greater impact

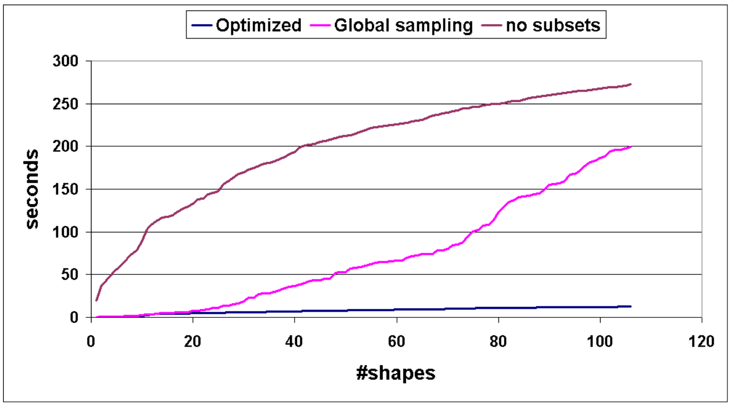


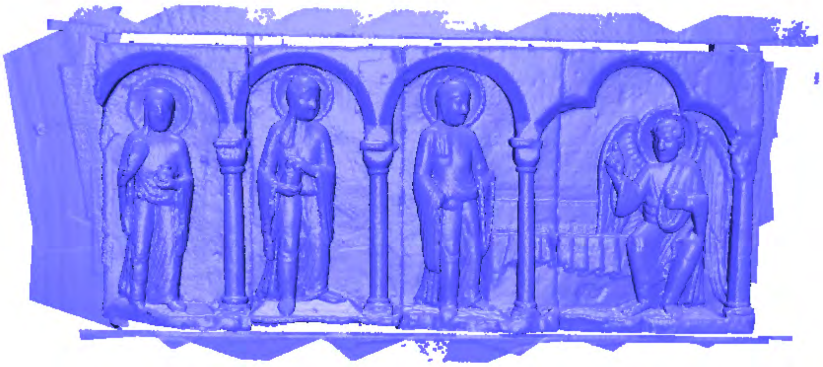
Figure 2.3: The chart shows the times of detection of the shapes found in the oil pump model when either subset evaluation or the localized sampling is disabled. For comparison also the timings of the fully optimized version are plotted. Total runtime for the version without subsets was 272.5s, 199.1s without local sampling and 12.3s with both optimizations activated.

early on in the process when the point-cloud and the shapes are both very large and thus many distance evaluations on octree cells and points have to be performed. In general the gain of the score evaluation on subsets increases with the size of the model. This way these two optimizations complement one another leading to significant performance gains at all stages of the detection.

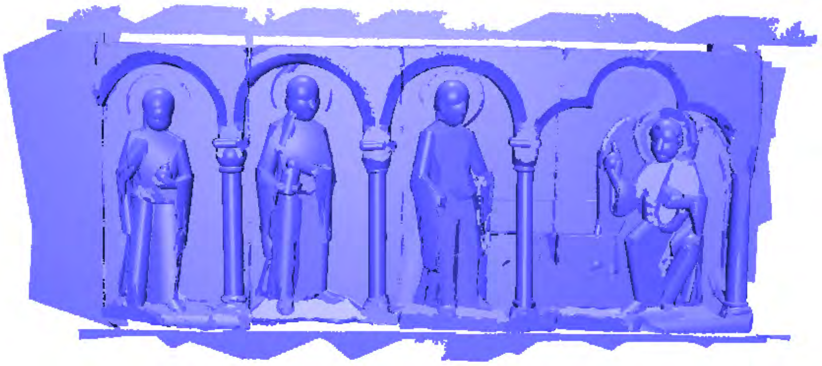
model	$\|\mathcal{P}\|$	ϵ	α	τ	$\|\Psi\|$	$\|\mathcal{R}\|$	sec
fandisk	12k	0.01	10	50	24	38	0.57
rocker arm	40k	0.003	20	50	73	1k	6.5
carter	546k	0.001	20	200	138	47k	29.1
rolling stage	606k	0.003	20	300	61	16k	15.1
oil pump	542k	0.0015	30	100	202	15k	30.9
master cyl.	418k	0.003	35	300	37	7k	12.1
house	379k	0.002	20	100	130	19k	10.7
church	1,802k	0.002	20	1000	160	690k	40.7
choir screen	1,922k	0.002	20	4,000	81	543k	20.8
				500	372	236k	61.5

Table 2.1: Statistics on processed models. ϵ is given as ratio of maximum bounding box width. Results have been averaged over 5 runs and rounded.

The choir screen in Figure 2.4 consists of a number of basic shapes, e.g. large planar areas, cylindrical or conical pillars, and more detailed parts such as the persons. The data was obtained with a laser range scanner and consists of several



(a) Original



(b) Approximation

Figure 2.4: The 372 detected shapes in the choir screen define a coarse approximation of the surface.

registered scans. In addition to some noise there also are some registration errors. The planar regions and the pillars are well detected but even for the persons good approximating shapes are found. Note that the small casks in the hands of the two leftmost persons are reliably detected despite their small size relative to the whole data set. The image was generated by projecting all points on their corresponding basic shape. For parameter values see Table 2.1.

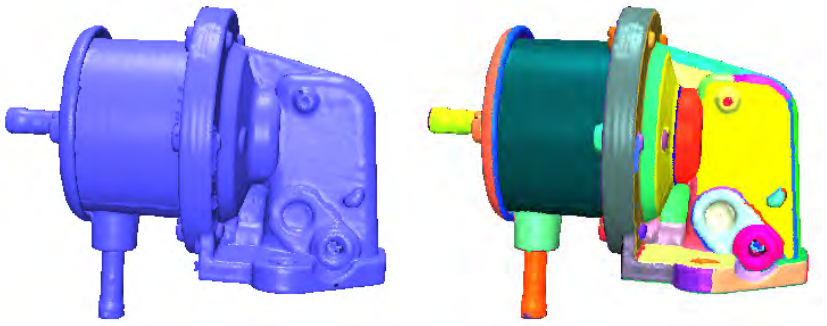
In contrast, the surface of the fandisk in Figure 2.2 is entirely composed of basic shapes. As expected, our algorithm is able to find these shapes and decomposes the surface into the constituting parts without leaving any remaining points. Note that the result is practically identical to that obtained by [WK05]. The oil pump on the other hand does not only consist of the basic shapes detected by our method. However, as is shown in Figure 2.5, existent basic shapes are well detected and only areas of blending patches or small scale details are ignored. The main characteristics of the model are captured concisely.

In Figure 2.7 additional results are presented. Outliers are ignored successfully and fine detail geometry is well recognized.

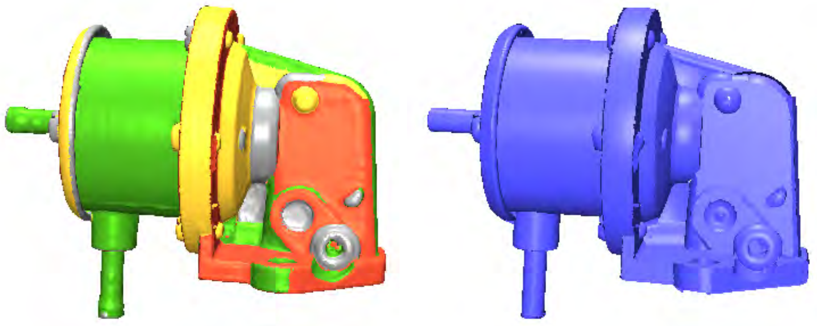
2.12.1 Noise

We have performed a simple experiment to demonstrate the ability of our method to handle noisy data. We use points of an octant of a sphere with increasing amount of synthetic gaussian noise and outliers as test cases. The points' normals have been obtained by locally fitting a least-squares plane to each sample's neighbors, and the radius in which the neighbors are collected is increased in accordance to the noise ratio. Note that no shape types have been deactivated during the detection, but we did not allow tori with a major radius smaller than the minor one because these form a superset of spheres. In Figure 2.9 an example with 10% noise and 80% outliers is depicted. Outliers are generated with a uniform distribution over the bounding box. Table 2.2 lists additional results together with the respective noise ratios. As can be seen, the original sphere parameters are reliably detected even for a large amount of noise as well as outliers. For a noise degree higher than 20% the detection starts to become unstable and the algorithm sometimes returns false shape types. Up to 95% outliers are tolerated for a noise degree of 2%.

Figure 2.6 b)-c) show the behavior of our algorithm on noisy data for a more complex model. The oil-pump model was distorted by synthetic Gaussian noise with σ equalling 1% of the bounding box diagonal. For such heavy noise we observe that only small or very narrow shapes (e.g. some of the screw heads) can no longer be reliably segmented since not enough support can be gathered for them. Another reason is that the estimated normals become too unreliable for these shapes, as the number of neighbor points used for the estimation needs to be

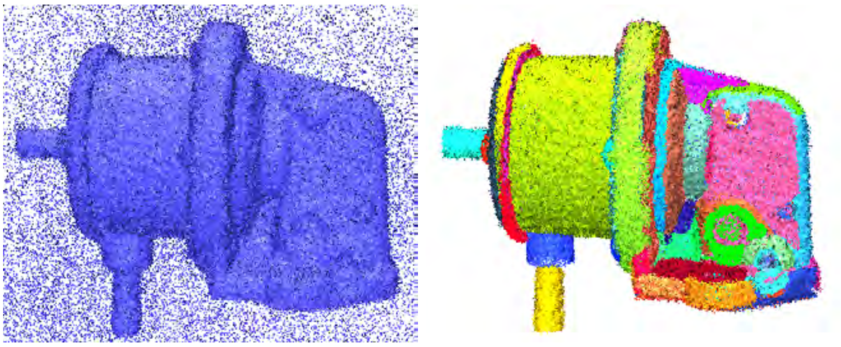


(a) Original (b) Random colors

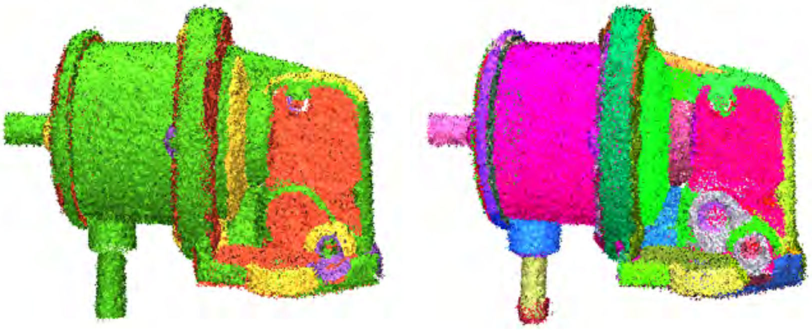


(c) Colored by type (d) Bitmaps

Figure 2.5: a) The original scanned model with ca. 500k points. b) Points belonging to shapes in random colors. c) Points of the shapes colored according to the type of the shape they have been assigned to: planes are red, cylinders green, spheres yellow, cones purple and tori are grey. No remaining points are shown. d) The bitmaps constructed for connected component computations provide a rough reconstruction of the object.



(a) Noisy original (b) Random colors



(c) Colored by type (d) Random colors

Figure 2.6: a) Distorted model with Gaussian noise and outliers b)-c) Results of the detection on the model with Gaussian noise but without added outliers. d) In addition to the Gaussian noise, 10% outliers were added (see a)).

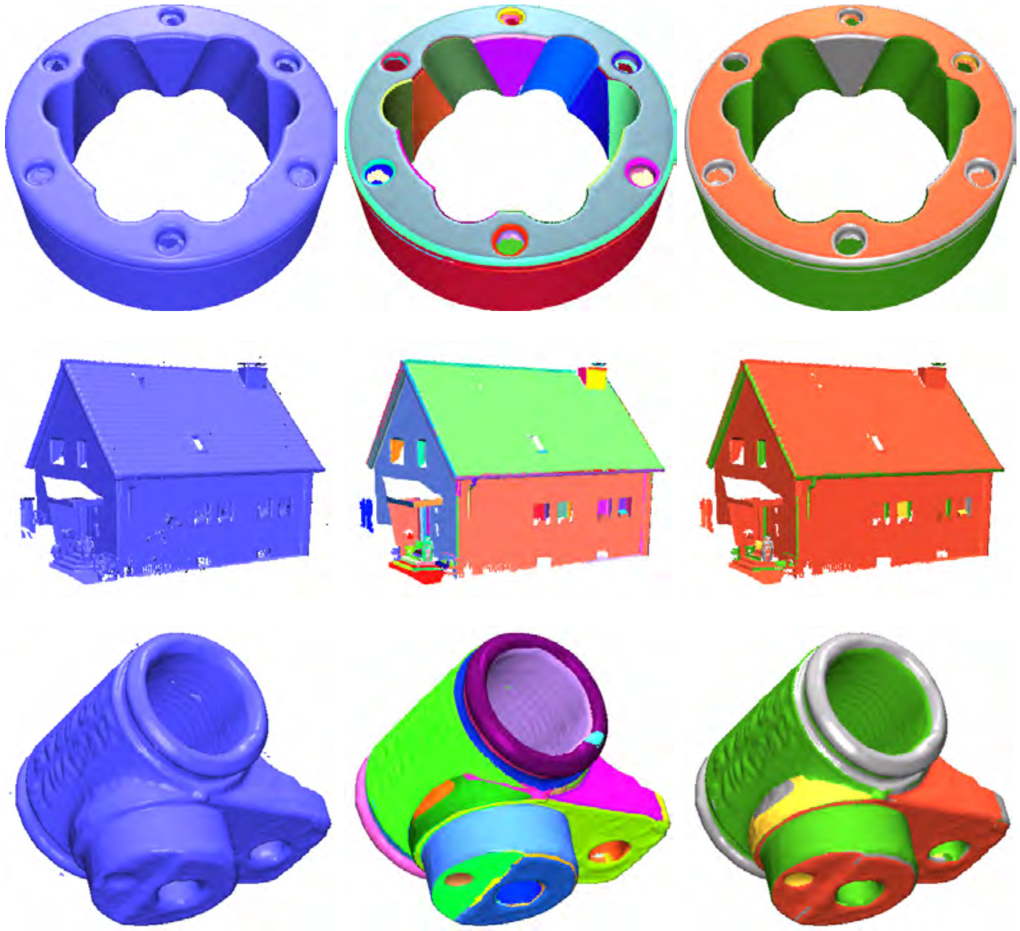


Figure 2.7: First column: Original point-clouds. Second column: Shapes colored randomly. Last column: Shapes colored by type as in Fig. 2.5. Models are from top to bottom: rolling stage, house, master cylinder. For parameters and timings see Table 2.1.

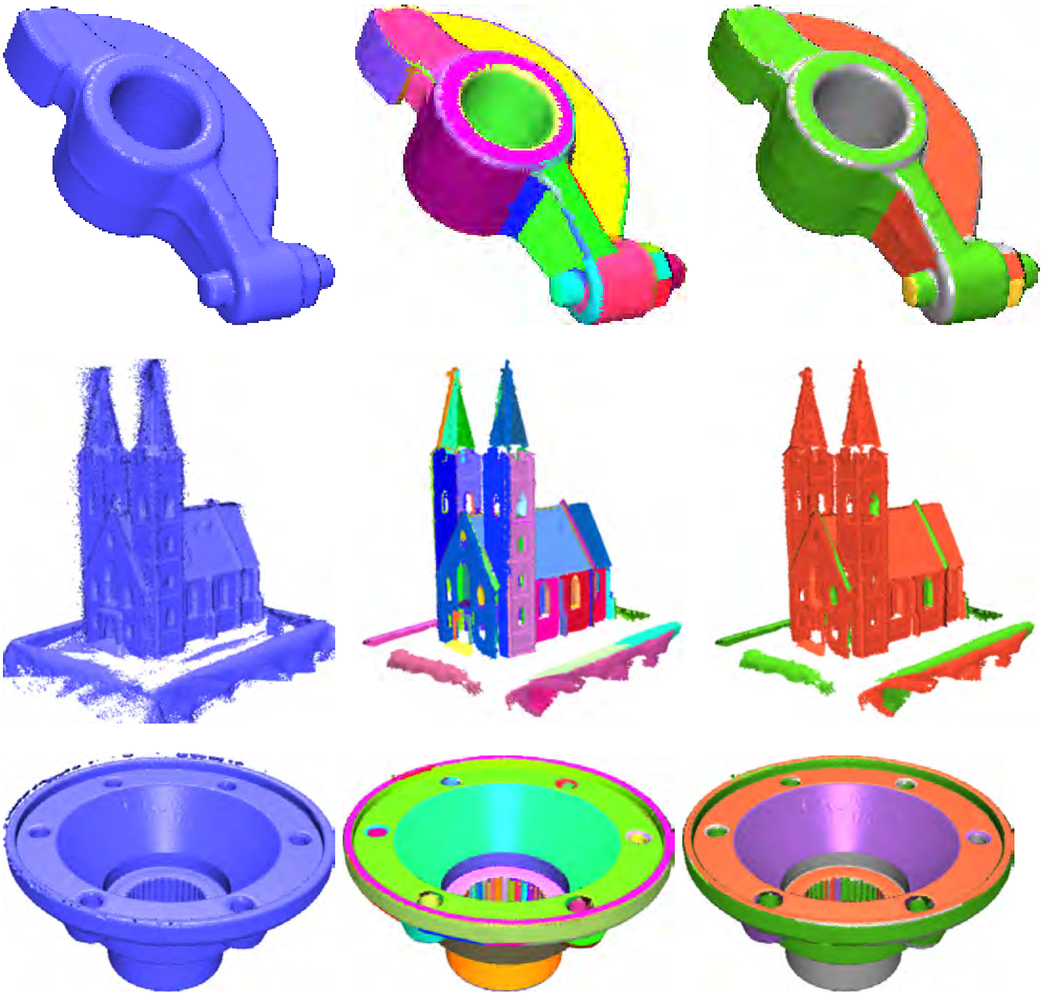
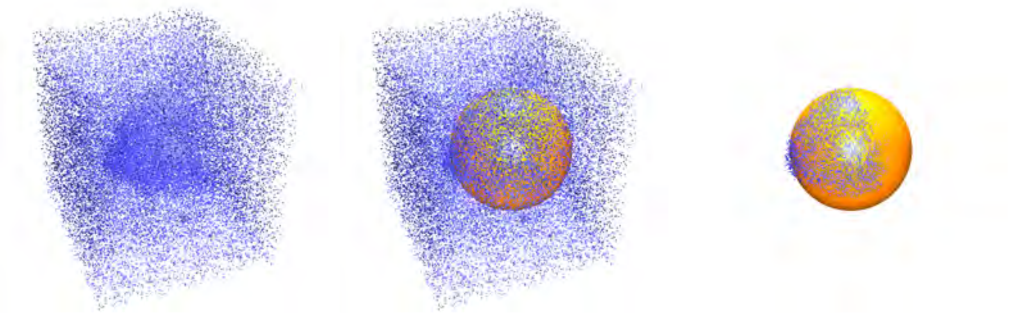


Figure 2.8: First column: Original point-clouds. Second column: Shapes colored randomly. Last column: Shapes colored by type as in Fig. 2.5. Models are from top to bottom: rocker arm, church, and carter. For parameters and timings see Table 2.1.

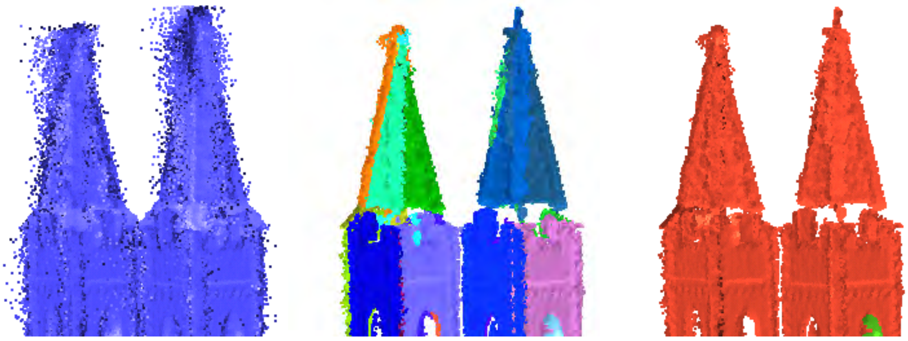


(a) Points (b) Sphere (c) Associated points

Figure 2.9: Points on an octant of a sphere distorted by synthetic gaussian noise with a σ of 10% relative to the sphere diameter and 80% outliers. Our algorithm is able to robustly detect the sphere, see also Table 2.2.

increased in order to smooth out the effect of the noise. Therefore points belonging to adjoining shapes adversely influence the estimated normals for such small shapes. The larger shapes, however, are unaffected and therefore are successfully and stably detected despite the heavy noise. In Figure 2.6 d) in addition to the noise 10% of the points have been randomly repositioned as outliers (again by a uniform distribution over the bounding box). As expected, the detection successfully ignores these outliers and the result is equivalent to that without outliers.

A real world example of heavy noise is demonstrated in Figure 2.10. Our method is able to successfully detect the planes comprising the roofs of the two spires despite the numerous outliers and noise artifacts.



(a) Original (b) Random colors (c) Colored by type

Figure 2.10: a) A part of the church model containing heavy noise. b) Points belonging to shapes in random colors c) Points colored by type of shape as in Fig. 2.5.

		before refitting				after refitting			
σ	o	μ_r	μ_c	σ_r	σ_c	μ_r	μ_c	σ_r	σ_c
0	0	0.02	0.02	0.02	0.02	0.00	0.00	0.00	0.00
1	25	1.73	2.24	1.16	0.95	0.07	0.07	0.004	0.004
2	25	1.79	2.60	1.43	1.47	0.31	0.31	0.021	0.023
5	50	2.25	3.96	1.65	2.19	0.35	0.26	0.054	0.025
10	50	8.11	11.87	4.15	4.44	4.32	7.20	0.53	0.91
10	80	7.17	10.58	5.18	4.68	5.12	5.99	1.65	1.96

Table 2.2: Parameter errors for the fitted spheres under different noise conditions compared to ground truth. All values are given in percent of the sphere diameter. The values are, from left to right, the level of gaussian noise σ, the percentage of outliers o, the mean values μ_r for radius and μ_c for center deviation, and the standard deviations σ_r of the radius and σ_c of the center.

Threshold	[BJ88]	[JB97]	our
0.7	16.65%	66.93%	71.06%
0.8	16.12%	66.50%	68.04%
0.9	16.14%	62.42%	62.35%

Table 2.3: Average percentage of correctly detected regions on the 40 test range images of the Segmentation Comparison Project. The threshold controls the ratio of required overlap between ground truth and machine segmented regions.

2.12.2 Comparison

Comparing our algorithm to other existing methods operating on 3D point-clouds is difficult. Most point-cloud segmentation algorithms do not explicitly detect instances of primitive shapes but find areas that are consistent in some other, predefined way. Moreover there are hardly any publicly available implementations or results. However, a benchmark for range image segmentation does exist (http://marathon.csee.usf.edu/seg-comp/SegComp.html)[PBJB98]. Therefore we resort to this benchmark to provide some comparisons of our method with other existing algorithms. Please note though, that the original intent of our method is not range image segmentation and the benchmark can therefore reflect only a subset of our method's abilities. The benchmark provides test images in conjunction with manually obtained ground truth segmentations as well as a tool for automatic evaluation. Due to the low quality of the used scanner the test images contain heavy systematic noise. Range image segmentation, of course, is a very special and simpler case of our far more general problem formulation that would allow for many extra optimizations and assumptions in the algorithm. Nonetheless we have tested our method *as is* on the test images provided by the benchmark.

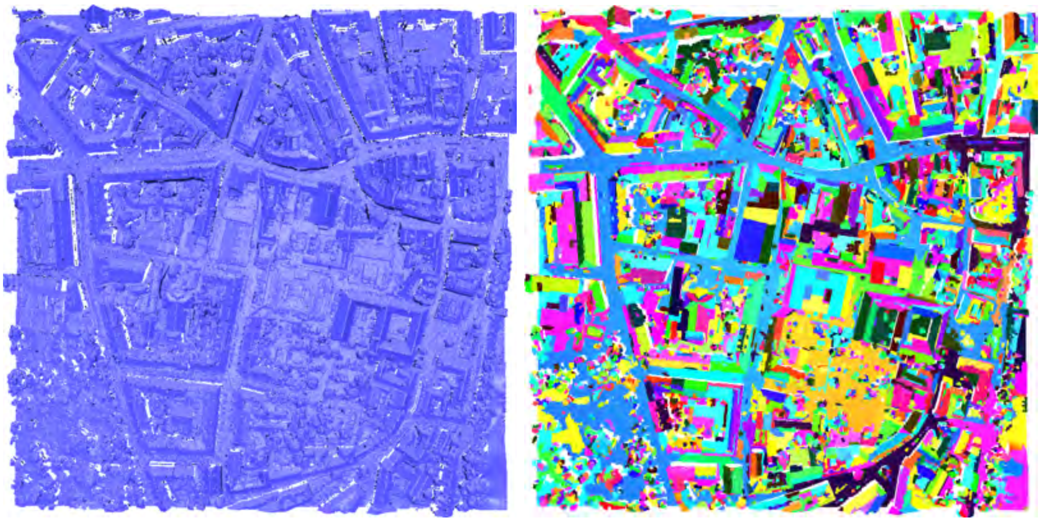


Figure 2.11: Primitives detected by the out-of-core version of the detection algorithm on 25 Mio. sample points from an aerial stereo reconstruction of Graz. The detection time was 470 sec.

Table 2.3 lists our detection results as well as those of the other two state-of-the-art methods for which results are available. Our method is able to achieve similar or even slightly higher rates of correctly detected regions as the UB segmenter [JB97], while clearly outperforming the BJ segmenter [BJ88]. The results are encouraging in the sense that our method can easily compete with these native range image segmenters without any further adaptations to this special case.

2.12.3 Out-of-core detection

We tested the out-of-core version of our algorithm on a part of the Graz dataset kindly provided by Heiko Hirschmüller from the DLR. The first subset we considered contains about 15 million sample points generated using the stereo reconstruction algorithm from aerial photographs described in [Hir08]. We restricted the maximum area of a shape to $1000m^2$ and the aspect ratio of an individual shape to $1 : 10$. We extracted 1120 shapes. About 3.5 million points, mainly belonging to other geometry like natural cover, were not assigned. The total extraction time was about 259 sec. For this medium sized data set we were able to compare this to our original algorithm which delivered a similar result with respect to the number of shapes and remaining points: 1000 shapes with 3.8 million remaining points were extracted in about 900 sec. Enhancing the area considered to a sub-set of about 25 million points of the Graz dataset and using our out-of-core algorithm we extracted about 2000 shapes with 5.7 million remaining points in about 470 sec (see Fig. 2.11), which is an approximately linear increase of runtime. The parameters were again a maximum area of $1000m^2$ per shape and an aspect ratio of $1 : 10$. A comparison with the original algorithm was not possible in this case due to the lack of main memory and the resulting disk-swapping. In summary we can state that partitioning the raw data set in parts that are adapted to the size of the sought primitives, enables the design of an efficient out-of-core RANSAC method for point-cloud shape detection that is especially well suited for huge city models. By adapting the size of the local subsets of the global point-cloud to the existent primitive shapes, the run-time complexity can be kept approximately linear in the overall number of points in the model.

2.12.4 Alternate score

In Fig. 2.12 results obtained with the simple score function of Sec. 2.7 and the MDL-based alternate score of Sec. 2.11 are contrasted. As expected, the MDL-based score finds more consistent types of primitives, i.e. similar parts in the point-cloud are represented by primitives of the same type. For instance, on the rolling-stage model (depicted in the bottom row) several cylindrical parts are represented by tori in the result obtained with the simple threshold-based score. These parts are all consistently approximated with cylinders in the MDL result. On the other hand, the MDL-score prefers primitives with fewer parameters. For instance, in the carter model (depicted in the top row) and also in the rolling-stage model, there are several spherical primitives in the MDL results that could arguably have been better classified as cylinders, cones or tori - as chosen by the simple score. This is the direct result of the trade-off between approximation quality and complexity offered by MDL which in this case prefers spheres as the additional approximation error caused by the spherical approximation is very small since the respective parts are quite narrow. For the rocker-arm model (depicted in the middle row) on the other hand this trade-off results in a clearly more reasonable choice of primitive types. Only on the tip on the lower right end there is a torus in the MDL result where a cylinder would have been more appropriate. A reason for this may be that the RANSAC algorithm can only choose from among the randomly generated candidates. If, by chance, no good cylinder could be generated then even the MDL score will select any other type of primitive that approximates the region. A possible approach to remedy this effect would be to apply a MDL-based post classification to each primitive. Indeed, in the case of the rocker-arm this does result in a cylinder. We proposed a more complete approach to this kind of post-processing that also considers the partitioning itself in Li et al. [LSSK09] and achieved promising results. Yet, the algorithm is not very efficient and is therefore not further discussed in this work where efficiency is a primary goal.

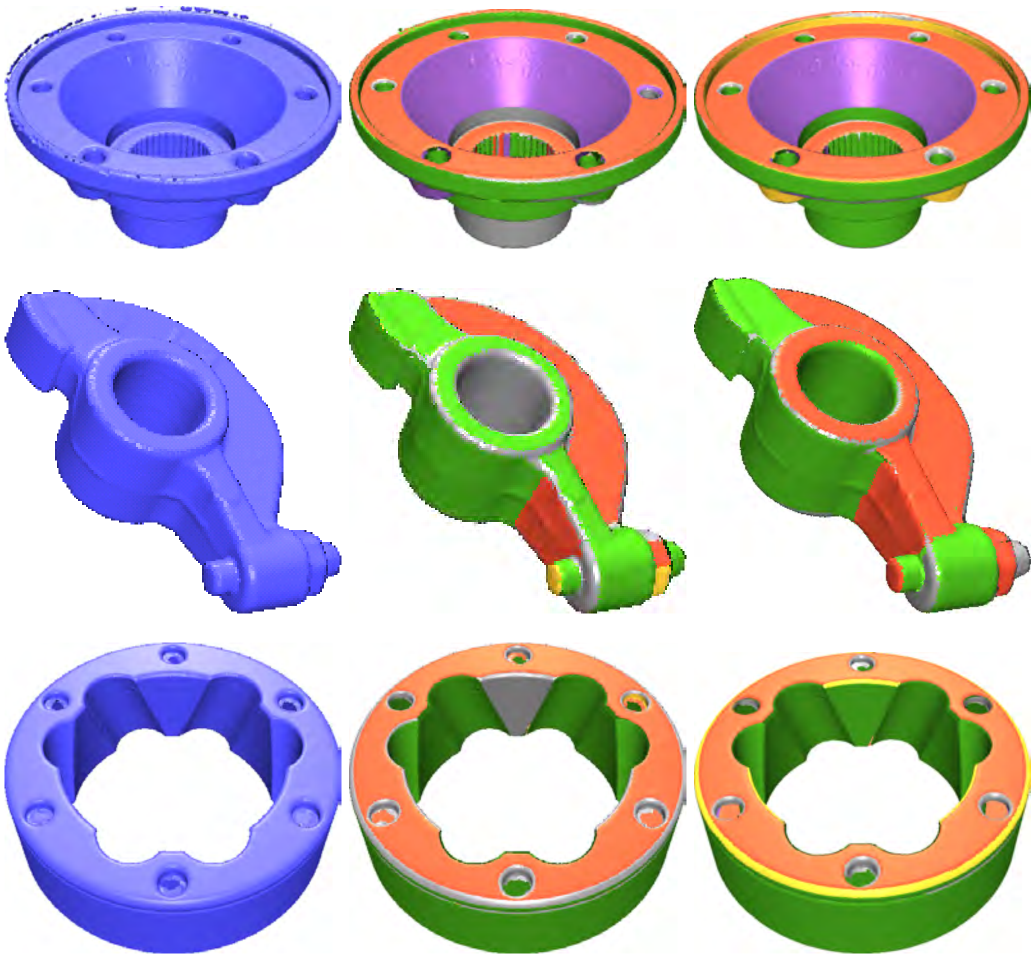


Figure 2.12: Comparison of results obtained with a simple thresholding score and the MDL based scoring approach. Left column: Input point-clouds. Middle column: Detected primitives with thresholding score, colored by type (cf. Fig. 2.7 and 2.8) Right column: Primitives detected with MDL score. Both approaches use the same distance and normal thresholds, bitmap resolution and minimum support. The weight for the parameters in the MDL score was set to 10. Note that the detected types are more consistent for the MDL score and in general primitives with fewer parameters are preferred.

2.13 Conclusion

After having outlined the potential in the well known RANSAC paradigm we have developed several fundamental extensions and derived new assessments for the probabilities involved therein for the case of shape primitive detection. We were able to present a randomized shape detection algorithm that is not dominated by the imponderability of chance, but is extremely robust and fast, and can be applied to a large variety of data from different sources and of different quality. Although our algorithm, in contrast to previous methods in computer graphics, does not find shape proxies for every part of the surface, we stress that this is not a requirement in many applications. The speed of the method, the quality of the results and its few requirements on the data render our algorithm a practical choice for shape detection in many areas, some of which will be explored in the following chapters.

CHAPTER 3

COMPRESSION

3.1 Introduction

As outlined in Chapter 1, one of the major challenges posed by scanned point-clouds is the sheer amount of data - on the one hand caused by the size of the acquired geometry but on the other hand additionally inflated by overwhelming redundancy due to non-adaptive acquisition. For these gigantic point-clouds, even seemingly simple tasks such as storage itself, let alone reconstruction and visualization, often pose a challenge. More importantly though, in spite of increasing bandwidth, it is still difficult to share these large datasets across a network. We believe that in this context high compression rates are of major importance, since we expect the size of datasets to grow much faster than e.g. hard-disk transfer rates in the future, while the increased computational demand of decompression is met well by the continuous and impressive gains in GPU processing performance. It is therefore desirable to have a format for this data that achieves very high compression rates while lending itself to quick decompression and rendering at the same time.

This chapter presents a novel algorithm that achieves just that: With the help of the detected primitives, the point-cloud is transformed into a compressed format using a novel fast vector quantization algorithm, such that the data can be decompressed in parallel on the GPU and rendered at interactive rates at the same time. For large models the compression rates of our scheme are similar to state-of-the art sequential point-cloud compressors. Besides geometry and normals we also show that our approach is applicable to colors as well. The main features of our algorithm are:

Bitrate We show that interactive high quality rendering at about 5-10fps on current hardware is achieved with *less than four bits* per input point including normals (see Fig. 3.1).

Normal estimation Our system allows trading compression of normals for image-space normal estimation. This way coding of point positions alone suffices

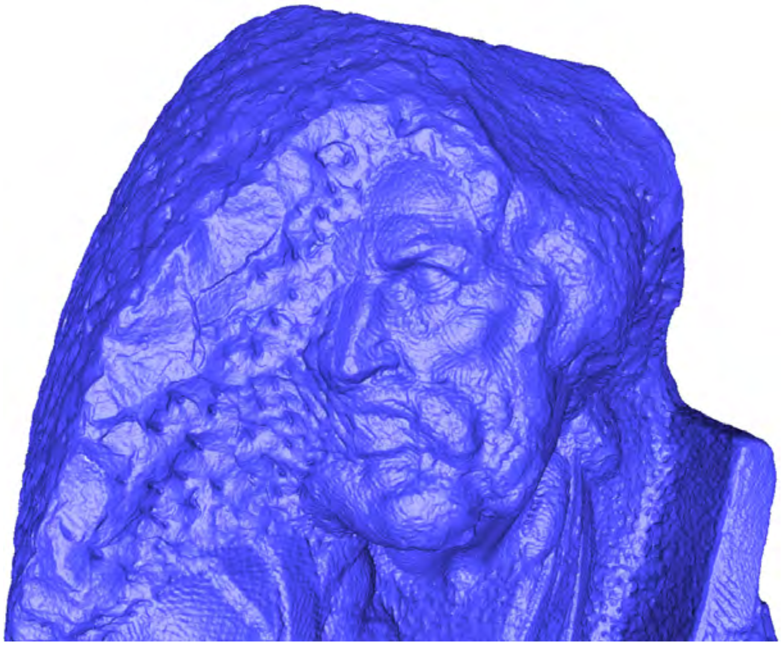


Figure 3.1: Michelangelo's St. Matthew rendered interactively at 3.31bpp including normals.

to obtain realistic renderings and point-clouds can be interactively rendered from *less than two bits* per point.

Level-of-detail Our method uses progressive compression and inherently supports level-of-detail.

Fast vector quantization We have developed approximative strategies for acceleration of the well-known Lloyd algorithm for vector quantization.

The compression is based on a decomposition of the point cloud into patches that are each approximated by a primitive shape, i.e. either a plane, sphere, cylinder, cone or torus (see previous chapter). The fine-scale geometry can then be encoded efficiently as height-fields over these patches. The height-fields are compressed progressively using image-based methods giving a level-of-detail representation.

The primitives provide a close approximation of the geometry for arbitrary models, but the primitive information can also be used to allow some user interaction, such as suppressing the rendering of selected items, moving or copying them. This is especially helpful in scenes such as buildings, cities or other man-made environments where the primitives are predominant and often are closely related to semantic entities. Moreover, in these scenes, the overall compression rate can be higher since the geometry is even more closely approximated by shape primitives.

3.2 Previous work

One of the first approaches combining compression with direct rendering was the QSplat system [RL00] which is based on a hierarchy of bounding spheres, giving a level-of-detail representation. The positions and radii (as well as other attributes) are delta coded in the hierarchy to reduce memory consumption. They require 6 bytes per input point with normal.

Botsch et al. [BWK02] use an octree as hierarchy for compression as well as rendering and sample the characteristic function of the surface into this representation. To encode the hierarchy in a coarse to fine manner, only the subdivisions of non-empty cells have to be stored in byte codes. This way they require less than 2 bits per leaf node. The number of leaf nodes is directly linked to the resolution of the finest octree level, since the characteristic function is continuous. Therefore, in general, more leaf nodes than original points are required to represent a model at its full precision, leading to much higher bitrates. To reduce the required grid precision, in each leaf, they store small offsets in normal direction, quantized to additional 2 bits.

Kalaiah and Varshney [KV05] use a statistical representation of the point-cloud to define a level-of-detail hierarchy. The hierarchy is constructed by computing a PCA of the point positions for each node and than dividing along the most significant axis. The parameters of the PCA, i.e. the local frame and the variances, are quantized to 13 bytes per node. Since leaf nodes represent a cluster of points, the hierarchy requires only about 8-9 bits per input point. The rendering of a node is based on quasi-random sampling of the encoded Gaussian distribution. By pre-computing a sequence of quasi-random numbers for a unit Gaussian distribution the sampling can be shifted to the vertex shaders of the GPU. The decoding of the node parameters however has to be done by the CPU.

Krueger et al. presented DuoDecim [KSW05], a point-cloud compression algorithm suitable for real-time GPU decompression. They resample the original point-cloud into a grid composed of Trapezo Rhombic Dodecahedra (TRD). Since a cell in a grid of TRDs has no second order neighbors, adjacency relations between cells can be encoded very effectively in only 2.25 bits. Thus, for compression of the grid, continuous runs of neighboring occupied cells are stored based on the simple adjacency relations. For decompression, several of these runs can then be processed on the GPU in parallel. The method achieves high compression rates of about 3bpp for positions and 5bpp for normals while introducing only a small sampling error. However, several grids have to be stored independently to obtain a level-of-detail representation.

In the context of raytracing Hubo et al. proposed two different algorithms for rendering from compressed point data. Their quantized kd-tree [HMHB06] allows decompression in depth-first manner which keeps the decompression overhead required during rendering low and is well suited for traditional raytracing acceleration schemes. However the achieved bitrates vary between 7 and 15bpp. In [HMHB07] Hubo et al. recently presented an alternative compression method also intended for raytracing applications. The approach is based on vector quantization of small surface patches, which allows direct and concurrent random access during raytracing. Bitrates as low as 1.4bpp are achieved, but result in noticeable smoothing of the surface. Visually appealing results are obtained at about 3bpp. Their algorithm is not suitable for GPU-based decompression as it does not support the fine-grained parallelism required for this architecture.

Compression of point-sampled geometry without having direct rendering in mind has also been studied extensively. Most related to our approach is the work of Ochotta and Saupe [OS04, OS08]. They partition the point-cloud into patches parameterizable over a plane. Similar to us, they resample the geometry as height-fields over these planes. Then they use progressive wavelet image compression on these height-fields to achieve bitrates of 2-3bpp for point positions. Other approaches in this area include the tree-based methods given in [HPKG06][SK06], the multiresolution algorithm provided by Waschbüsch et al. [WGE+04] and the single-rate coders of Gumhold et al. [GKIS05] and Merry et al. [MMG06]. While all these methods achieve high compression rates, in contrast to our approach, decompression is sequential and cannot be performed on the GPU for interactive rendering. Nonetheless our algorithm is able to achieve similar bitrates on large models.

The height-field geometry representation employed in our system is a well known concept that, besides its use in compression (see above), spectral analysis [PG01], simplification and reconstruction [BHGS06], has also been used for rendering. Ivanov and Kuzmin [IK01] propose the use of planar range images as rendering primitive in a hardware pipeline. However, they use a large number of very small patches and do not consider any compression. Ochotta and Hiller [OH06] designed a rendering system based on height-fields that achieves high quality renderings at interactive rates. However, they too, do not consider compression or level-of-detail.

3.3 Overview

Our method is an asymmetric compression algorithm that is based on vector quantization of the height-fields' Laplace pyramids. The first step is the decomposition of the input point-cloud into parts parameterizable over a primitive shape as described in the previous chapter and depicted in Fig. 3.2 a).

Once the decomposition has been obtained, the geometry will be represented as an atlas of height-fields that have been resampled on a regular grid located in

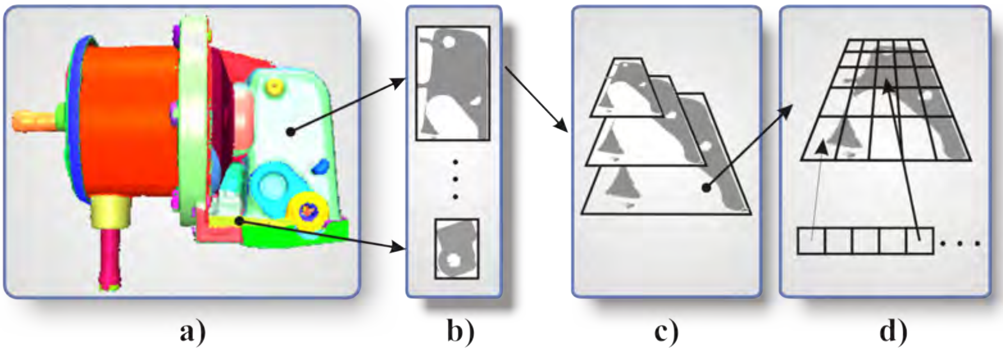


Figure 3.2: Different stages in our compression algorithm. a) The object is decomposed into parts corresponding to shape primitives. b) Height fields over the primitives are generated to describe fine scale details. c) Laplace pyramids are computed for each height field d) Pyramid levels are encoded with vector quantization.

the domain of the respective primitive [PG01][OS04] (Fig. 3.2 b) and Sec. 3.4.1). Note that these height-fields are allowed to have irregular boundaries and we store with each field a binary occupancy mask to encode the existence of surface samples in the corresponding grid cells. Since height-fields are basically equivalent to grey-scale images we also refer to the height-fields as images throughout this work.

Each height-field is filtered and downsampled to yield a collection of equally high image-pyramids (Fig. 3.2 c) and Sec. 3.4.2). Starting with the coarsest pyramid level, the images of all shapes are simultaneously vector quantized. The quantized versions are then upsampled and subtracted from the next finer resolution images, resulting in difference images. Such difference images are then successively vector quantized for each pyramid level (see Fig. 3.2 d) and Sec. 3.4.3).

Decompression then simply replaces codebook indices with codebook vectors and sums up as many difference images as required to reconstruct a selected pyramid level. This can be done efficiently on the GPU using dependent texture lookups (see Sec. 3.5). The resulting height-fields are then reconverted into point-clouds for rendering.

During rendering, level-of-detail is realized by choosing a pyramid level for each patch such that the level's resolution guarantees a hole-free point rendering. Should a hole-free rendering require a higher resolution than available, a lower resolution framebuffer is chosen as render target. These lower resolution images are later scaled and merged into the target resolution to obtain the final rendering.

During this image processing, normals can also be generated from the stored depth information (see Sec. 3.6).

3.4 Compression

After the shape primitives have been obtained, the height-field atlas has to be generated. To this end, the point-cloud is resampled on a regular grid in the domain of each shape. Recall that the compressed point-cloud will consist only of these resampled positions, so that care has to be taken to avoid visible gaps between the edges of different shape primitives. As was observed by Ochotta and Hiller [OH06], in order to obtain hole free rendering, it suffices to have the patches slightly overlap.

3.4.1 Resampling

The result of the resampling is a regularly sampled height-field (or grey-scale image respectively) together with a binary mask specifying the valid entries. The height-fields are sampled in the parametrization domain of the respective shape primitive. To establish the location and extent of the resampling grid in the parametrization domain for each patch, all points assigned to a shape are projected onto the respective primitive and a bounding box is found for all these projected points in the parametrization domain. In order to preserve the original number of points, the resolution of each resampling grid should be chosen such that the number of occupied cells equals the number of points in the respective patch.

We find the height-field's resolution and the binary mask of occupied cells in a joint iterative process. The initial resolution of the grid is set to the average distance between a projected point and its nearest neighbor. In each iteration the projected points are sorted into the grid and a morphological closing operation is used to fill small holes in the resulting occupancy mask. Then, similar to a binary search, if the number of occupied cells after the closing is larger than the original number of points, the resolution is set to the middle value between a lower bound and the current resolution. Otherwise the resolution is increased analogously. After the correct resolution has been determined, the constructed binary mask is dilated once to ensure a slight overlap between neighboring patches in space.

Heights Now, for each occupied cell a height value has to be computed. These height values are obtained by intersecting a ray in direction of the shape primitive's normal with the moving least-squares (MLS) surface (see Sec. 1.5.2) for each masked cell [AA03b]. Using the MLS-surface has the advantage that different patches use a consistent surface definition across patch borders, which ensures that no edges will be visible in the resampled point-cloud.

Normals In addition, the MLS surface can also be used to obtain approximate normals, which can be stored along with the offset value if desired. Point normals are encoded relative to the primitive's normal using spherical coordinates. Using the primitive's normal as reference results in a low entropy for the polar angle, as it will usually be close to zero.

Colors If the points are equipped with colors, the MLS-surface is also used to resample the colors. For compression, colors are converted from RGB to YCrCb, so that higher precision can be used for the intensity channel.

3.4.2 Filtering

The acquired height-fields are successively filtered and subsampled to obtain image pyramids. The levels of these pyramids constitute the levels-of-detail supported by our method. A level P_i of the pyramid is obtained as

$$P_i = \downarrow g(P_{i-1}), \tag{3.1}$$

where g is a low-pass analysis filter and $\downarrow$ denotes subsampling. P_0 is the original image. Rather than storing the pyramid levels independently we use the Laplacian pyramid representation introduced by Adelson and Burt [AB81] to achieve better decorrelation (and thus compression). A level of the Laplacian pyramid is the difference between the corresponding level of the image pyramid and the upscaled lower resolution level. Thus a level L_i of the Laplacian pyramid is given as

$$L_i = P_i - h(\uparrow P_{i+1}), \tag{3.2}$$

where h is a synthesis filter and $\uparrow$ denotes upsampling. Only on the coarsest level l, Laplacian and image pyramid are identical, i.e. $L_l = P_l$. We can then reconstruct a level P_i from the Laplacian pyramid by recursively applying

$$P_i = L_i + h(\uparrow P_{i+1}). \tag{3.3}$$

Originally, Adelson and Burt suggested to use Gaussian-like filter kernels for g and h in the pyramid construction. However, Gaussian analysis filters also require Gaussian synthesis during reconstruction of pyramid levels. With a GPU implementation of the reconstruction in mind, we use the CDF 5/3 [CDF92] wavelet instead, as in this case the low-pass synthesis filter simply is a bilinear interpolation, which is natively supported in the hardware.

3.4.3 Vector quantization

Vector quantization works by replacing small tiles of the original image with indices into a codebook [GG92]. The codebook simply contains a set of representative tiles. The main reason to use vector quantization in our method is that this simple scheme directly lends itself to parallel decompression while achieving high compression ratios. To obtain the decompressed image all indices can be replaced with the vectors from the codebook independently and concurrently. In principle, all this amounts to are dependent texture look-ups on the GPU.

Several related approaches have also used vector quantization to enable fast decompression. Beers et al. [BAC96] introduced the concept to the computer graphics community in the context of texture compression and Levoy and Hanrahan used vector quantization for Light field rendering [LH96]. In [SW03] Schneider and Westermann showed that the scheme can also be applied to compression and rendering of volume data.

Thus, to achieve compression we apply vector quantization to the pyramid levels L_i [HH88]. For the vector quantization the height-fields are decomposed into small vectors corresponding to square tiles of side length x. All the vectors are collected in a set $\mathbf{V}$ of data-vectors. Each data-vector carries a binary mask, that has been generated from the occupancy bitmap, to identify any missing values. No vectors are generated for empty regions in the patch.

The key to high compression rates is to find a small codebook together with a mapping from original tiles to code-vectors such that the distortion introduced by the replacement is minimized. Let $\mathbf{C} = \{c_i, \ldots, c_k\}$ be a set of code-vectors and $\mathbf{V} = \{v_i, \ldots, v_N\}$ be the set of data-vectors (or image tiles respectively), then distortion is measured as the root mean square (RMS) error:

$$R = \sqrt{\frac{1}{N}\sum_{i=1}^{N} m_p(v_i, c_{M(i)})^2}, \tag{3.4}$$

where $M : \mathbb{N} \rightarrow \mathbb{N}$ is the mapping assigning data-vectors to code-vectors and m_p is a Minkowski metric. In our case m_p has to respect the missing values in some of the data-vectors in V. Thus, we use

$$m_p(x, y) = \sqrt[p]{\sum_{i=1}^{d} b_i^x b_i^y |x_i - y_i|^p}, \tag{3.5}$$

where $b^x \in \{0, 1\}^d$ denotes the binary mask associated to vector x. Although different choices would certainly be possible, for simplicity, we always use the standard least squares error metric, i.e. $p = 2$.

In our system the user specifies a maximal RMS error R_{max} prior to compression, such that a codebook and mapping have to be found accordingly. While we treat different pyramid levels independently, on each level we use a common codebook across all shape patches, i.e. the data-vectors $\mathbf{V}$ are collected on the respective level from all patches in the atlas. We do not quantize across scales for two reasons: Firstly, we can avoid accumulating quantization errors if we compute the levels of the Laplace-pyramid using the previously quantized lower resolution images. Secondly, any codebook across scales has to be large enough to achieve an error less than R_{max} on L_0. This leads to an overly verbose codebook on coarser levels and, consequently, to large code-vector indices requiring many bits.

3.4.4 Codebook generation

We base the codebook generation on the LBG-Algorithm [LBG80]. It is well known however that a naïve implementation of this method exhibits extremely poor runtime performance. Here we will describe several effective acceleration methods, some of which we believe have not been described previously, and achieve a runtime improvement of several orders of magnitude compared to the unmodified algorithm with only marginally increased error.

LBG-Algorithm The Lloyd algorithm finds an optimal mapping M by assigning each data-vector to its nearest code-vector, i.e.

$$M(i) = \arg\min_{j=1\ldots k} m_p(c_j, v_i), \tag{3.6}$$

where k denotes the number of clusters. Optimal code-vectors are then computed as the centroid of all data-vectors which they are to represent.

$$c_i = \frac{1}{|M^{-1}(i)|} \sum_{j \in M^{-1}(i)} v_j, \tag{3.7}$$

where $M^{-1}(i)$ denotes the set of data-vectors assigned to c_i. This is repeated until convergence. Building on Lloyd's algorithm, the LBG-Algorithm also addresses the problem of finding the necessary number of clusters k. Starting out with a given set of cluster centers, the LBG-Algorithm executes Lloyd's algorithm and if the required error R_{max} has not been achieved, inserts additional clusters and iterates. The new centers are found by "splitting" of old cluster centers, i.e. by adding small opposing random offsets to the old center.

A large body of work deals with the efficiency and scalability of the LBG-Algorithm (also referred to as k-Means algorithm) in the context of vector quantization as well as clustering. Many efficient techniques concentrate on accelerating the nearest neighbor queries necessary for finding the optimal mapping in every iteration of the algorithm with geometric reasoning or data structures [KMN$^+$02, ARS99, PM99, KR93, PH98, BG85, Elk03]. To apply these methods in our case they have to be extended to deal with missing values. Other methods reduce the number of vectors that have to be inspected by either sampling [KGKB03, FS06a] or by summarization [ZRL96, BFR98, GRS98, OLY05]. Both sampling and summarization work best if the data contains strong natural clusters and the number of sought cluster centers k corresponds to the number of these natural clusters, as is often the case in data-mining applications. In our setting however, the number of natural clusters usually is far less than the number of required code vectors because the natural clusters contain too much variation for the sought error threshold R_{max}. Consequently, sampling or summarization alone do not suffice in our case. Thus we combine geometric data structures, sampling and approximative evaluations to achieve the desired acceleration.

Filtering We choose to accelerate the nearest neighbor queries with a filtering approach that is based on a kd-tree like structure containing the data-vectors, because of its effectiveness and simplicity. We augment the approach of Alsabti et al. [ARS99] with respect to missing values. The method constructs a kd-tree on the data-vectors. The nearest cluster center for each data-vector is then found in a top-down traversal of the tree. During the traversal a list of active cluster centers is maintained for every subtree. The list of active centers is pruned at each node of the tree. To this end the minimal and maximal distance of each cluster center to the node's corresponding cell is computed. All centers with a minimal distance larger than the minimum of the maximal distances are deleted from the list and the traversal continues on the child nodes with the remaining clusters only. Once a leaf is reached or the number of centers in the list drops below a small constant, the nearest neighbor is found among the active centers for all data-vectors of the current node in a brute-force fashion.

In case of missing values in the data-vectors two kinds of axis-aligned bounding-boxes are necessary to compute the minimal and maximal distances for each node: B_{min} is used to find the minimal distances and B_{max} for the maximal distances respectively. B_{min} has infinite extent in all dimensions where missing values occur in the data-vectors. On the other hand, to obtain B_{max}, all missing values simply are ignored (in an axis for which no data-vector has a valid entry, the bounding box is marked invalid as well). This way correct minimal and maximal distances can be obtained for each node of the tree, see Fig. 3.3.

However, a problem still arises during the construction of the tree. Each node of the tree splits its corresponding cell into two parts along a plane orthogonal to a chosen axis. Points on the left or right side of the plane are sorted into the left or right child node respectively. But what happens if a data-vector's value on

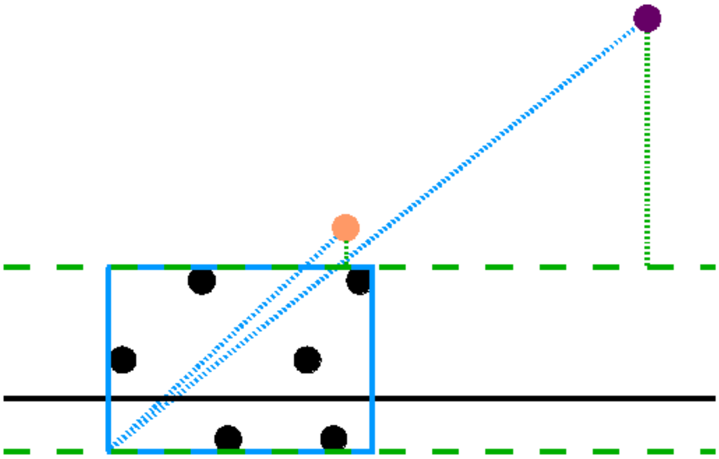


Figure 3.3: Kd-tree filtering with missing values. Black points depict vectors without missing values. The black line depicts a vector with a missing value on the x axis. The bounding box B_{max} is shown in blue, the infinite box B_{min} is shown in green dashed lines. The violet cluster center is pruned because its minimal distance to B_{min} is larger than the maximal distance of the orange cluster center to B_{max}.

the chosen axis is missing? These points could be sorted into either child node, but there they would keep causing infinite extent in B_{min}, which might hinder effective pruning of centers. Thus we introduce a third child node into which we sort all data-vectors with a missing value in the chosen axis. Then a subtree is constructed recursively for all child nodes. Please note that more effective kd-tree filtering techniques have been proposed in [KMN⁺02] and [PM99], but cannot be adapted to the case of missing values in the data-vectors because they rely on finding the midpoint of a node's cell which is undefined if B_{min} has infinite extent.

In every iteration of the LBG-Algorithm new cluster centers are introduced. It is wasteful to consider all cluster centers to recompute the nearest neighbor of the data-vectors since only the new cluster centers have changed. By keeping track of the modified clusters and the distance to the previous nearest cluster of each data-vector, unnecessary distance computations can be avoided in the search.

Approximate Lloyd Even with the kd-tree filtering in place, the nearest neighbor queries still constitute the limiting factor in the runtime of the Lloyd algorithm. Based on the observation that a data-vector is unlikely to be reassigned to a cluster center which was not among its l nearest neighbors in the first iteration, we suggest to replace exact nearest neighbor queries on the entire set of centers with nearest neighbor queries on tiny, precomputed subsets of centers, one for each data-vector respectively. These subsets are chosen as the l nearest centers of each data-vector during the first iteration of the Lloyd algorithm. Since these nearest neighbors are found in the first iteration and then kept fixed, the true nearest center of a data-vector may not always be found later on if centers should move wildly. However, the observed error is extremely small.

We further propose an additional approximation, which is able to eliminate about half of all distance computations in our experiments. In the algorithm the lists containing the nearest neighbors of each point are initialized with the kd-tree filtering approach described above, which can trivially be extended to find the l nearest neighbors of each point. Just as was the case previously, in the following iterations of the algorithm nearest neighbors are then found only from among these lists respectively. However, all points whose distance to their second nearest neighbor is less than $1 + \epsilon$ the distance to their nearest neighbor are ignored, i.e. they are always kept assigned to the cluster center which was nearest when the list was created. We can expect the number of points for which this condition holds to be a large fraction of the total number of data-vectors (and therefore we can also expect an effective acceleration). This is due to the fact that, for higher dimensions, the ratio of the distance between the nearest neighbor and the farthest neighbor tends towards one. For a proof see e.g. Beyer et al. [BGRS99] or Aggarwal et al. [AHK01]. A more graphic explanation is obtained if we take a look at the ratio of volume between a sphere of radius r and $(1 + \epsilon)r$ which is zero in the limit for infinite dimension d:

$$\lim_{d\to\infty} \frac{\pi^{\frac{d}{2}} r^d}{\Gamma(\frac{d}{2}+1)} \cdot \frac{\Gamma(\frac{d}{2}+1)}{\pi^{\frac{d}{2}}((1+\epsilon)r)^d} = \lim_{d\to\infty} \frac{r^d}{(1+\epsilon)^d r^d} = 0 \tag{3.8}$$

Thus we can expect the second nearest neighbor to fall into the volume of the ϵ envelope of the sphere enclosing the nearest neighbor with high probability. The reason this approximative strategy introduces only small error is that these points are very close to the bisector separating the Voronoi cells of their two nearest neighbors, which causes them to be frequently reassigned to either of these centers. But the reduction in total error gained by this reassignment is only negligible (probably less than $\frac{1}{1+\epsilon}$). Thus keeping their assignment fixed introduces only a small additional error but increases significantly the convergence rate of the Lloyd algorithm, while at the same time eliminating a large fraction of the overall required distance computations. However, in order to avoid prematurely fixing cluster centers in their locations, we do reassign all vectors every 10 iterations. In all our experiments we set $\epsilon = \frac{1}{10}$.

Subsets Even though, in our setting, working on a subsampled set of data-vectors usually does not allow us to identify all cluster centers necessary to achieve

Algorithm	Time (sec)	bpp
BF	398.59	3.40
BF/S	109.48	3.40
F	236.26	3.40
F/S	68.39	3.40
F/L	90.53	3.41
F/L/S	12.39	3.41
F/L/E	19.77	3.42
F/L/E/S	7.37	3.41

Table 3.1: Effects of the different acceleration methods when applied to the oil-pump model (see Fig. 3.2). BF denotes brute force nearest neighbor search, S subsampling, F filtering with kd-tree, L the search in the list of l nearest neighbors and E ignores points close to a cluster bisector. L and E are approximating strategies, but bitrates are effected by less than 1%. The required time for construction of the kd-trees for the final F/L/E/S algorithm is 0.6sec (included in the overall timings above).

the desired tolerance R_{max}, starting the clustering on small subsets often helps to quickly find good initial cluster positions and may even be able to identify tight natural clusters early on. Therefore we incorporate sampling into our approach in a simple but effective manner: We start with a small random subset of the data-vectors on which we execute our codebook generation with the user specified R_{max}. We then double the size of the subset by including additional random data-vectors and restart the codebook generation using the cluster centers obtained in the previous run for initialization.

Results In Table 3.1 timings for different combinations of the described acceleration methods are listed. Each time the same error threshold R_{max} was used and bitrates are computed from the written compressed file, i.e. they include codebook vectors as well as shape primitive information. We tested the different variants on a relatively small model (the oil-pump of Fig. 3.2) in order to be able to get timings even for the inefficient strategies, e.g. the brute force search. On larger models the acceleration factors are even higher.

Normals and colors Each normal is encoded relative to the shape primitive's normal with two angles ϕ and θ (see Sec. 3.4.1). We found that higher compression is achieved if, instead of storing ϕ and θ in a single vector, two vectors, one for ϕ and one for θ, are generated for each square tile and then a single codebook is generated for both kinds of vectors. This is due to the lower dimension of the separate vectors as well as the lower entropy of the θ vectors since it tends to be close to zero.

In the case of colors we also generate two vectors for each image tile. One vector contains the Y component, while the other vector contains both, the Cr and Cb components. Again we achieve higher compression by this separate handling of components. In the case of colors however it also makes sense to apply a smaller threshold R_{max} to the vector quantization of the CrCb vectors since the human eye is less sensitive to errors in these channels than in the Y component.

Scalar quantization After the codebook generation, the elements of the code-vectors additionally undergo scalar quantization into eight bits per element. This way they can be stored in single component textures on the GPU.

3.4.5 Hierarchy

In principle, it is possible to store the compressed pyramid levels as two dimensional arrays of code-vector indices. However, this would imply saving indices even in empty regions of a level. Since the occupancy masks may indeed contain large empty areas, this wastes a lot of space with useless information. Thus it is significantly more efficient to use a quad-tree representation of the pyramid, in which only the occupied areas have to be stored.

Quad-tree

Traditionally, each node of the quad-tree would contain a code-vector index together with a list of pointers to the existing child nodes. The list of pointers can be replaced by four bits specifying the existing children if the quad-tree nodes are stored in breadth- or depth-first order. However, during decompression we want to be able to process many quad-tree nodes in parallel on the GPU, and while such a representation is very space efficient, it is not well suited for parallel processing due to the sequential nature of the traversal order. Thus, in the spirit of the well-known recursive data pattern [MSM04], to enable efficient parallel decompression we store instead a pointer to its parent together with two bits specifying its child relation. We keep the levels of the quad-tree in separate arrays, such that the pointers actually are offsets into these arrays. This representation allows us to process each node on a level in parallel with all other nodes of the same level at the cost of additional pointers in the leaf nodes.

Parent pointers The encoding of parent indexes in the quad-tree nodes may become problematic if the images are very large, and therefore indices into large

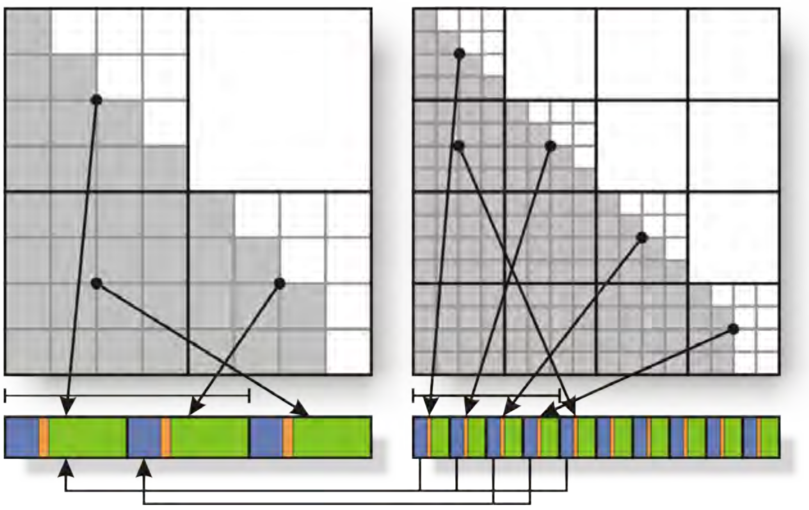


Figure 3.4: Two consecutive levels of an image quad-tree. Each quad-tree cell contains x^2 pixels. Below the quad-tree tiles the array for the level is depicted. Each entry stores a pointer to the parent tile, the child relation and the code-vector index. Array entries corresponding to partial tiles, i.e. tiles with incomplete occupancy masks, are sorted to the beginning of the array.

arrays would have to be stored. Thus, to avoid spending to many bits on the parent indices, we subdivide every image into square blocks of side length $2^l x$ (recall that l is the number of pyramid levels and x is the side length of the quantized image tiles). This limits the number of bits required to store parent offsets to $2(l-1)$ on the finest level. Moreover, as a side effect, we can use these sub-blocks to define the granularity of our level-of-detail selection. To this end, we also store a bounding box along with each block.

For a compression using five pyramid levels and using a tile side length of four the expected bits per point required on the last level for the parent indices can be now be computed as $\frac{2(l-1)}{x^2} = \frac{1}{2}$. This is still a significant amount which we can further reduce by sorting the tiles of the last and next to last level such that nodes that have four children are stored in an order which allows implicit quad-tree indexing of the parent (and therefore of the child relation as well). This way we usually save more than 50% of the overhead caused by parent indexes on the last level.

Occupancy bitmaps The number of bits needed for a node of the quad-tree now depends on the number of bits required to encode a code-vector index, the offset into the parent array and two bits for the child relation. Unfortunately however, it does not suffice to store the code-vector index alone for decompression, as some of the quad-tree cells may only be partially occupied. For these cells a bitmap is used to encode the occupancy. Please note that we call a quad-tree node partial if not all of its associated image pixels are occupied, which is independent of the number of children of the node.

To minimize storage overhead for the occupancy bitmaps, all partial quad-tree tiles of each level are sorted to the beginning of the level's array and the corresponding bitmaps are stored in the same order in a second array. Fig. 3.4 illustrates the resulting layout of the data structure. This way, the only overhead is the number of partial nodes that has to be encoded for each level and no additional information is needed for full nodes.

3.4.6 On-disk compression

The compressed format as described up to now is suitable for direct decompression on the GPU as will be described in the following section. However for storage on disk, further compression can be achieved using entropy coding. Entropy coding, especially arithmetic coding, is hard to parallelize due to the varying number of bits per symbol as well as the serial model updates in an adaptive coder [You98]. However, adaptivity is very important in our setting to achieve notable compression gains. Therefore arithmetic coding [RL79] is used to store the data on disk and is decompressed on the CPU when the object is loaded to memory. The parallelly decodeable format is kept in main memory for upload to the GPU.

The codebooks for each level are compressed using simple adaptive arithmetic coding [WNC87]. For the occupancy bitmaps we employ context adaptive arithmetic coding, where the context is defined by the causal neighborhood of each entry. To compress the hierarchy, i.e. parent pointers, child relations and code-vector indices, we have developed two different variants. In the first one all entries of the quad-tree arrays are compressed with adaptive arithmetic coding. This is very simple and can be decompressed during loading with practically imperceptible overhead. In the other variant the quad-tree structure is serialized in a breadth-first manner, so that no parent pointers are required. This obviously achieves higher compression gain but also causes notable overhead (which however is still uncritical) during decompression due to the necessary reordering of quad-tree nodes and generation of parent pointers. Table 3.2 lists the results of the different compression modes when applied to a set of test models. On-disk compression achieves improvements between 15% to 40% compared to the GPU decodeable format.

3.5 Decompression

The aim of our compression technique is to allow for fast decompression on the GPU, which has two advantages: Firstly only the compressed data has to be sent across the bus to the GPU during rendering and secondly the high degree of parallelism of the GPU's SIMD structure can be fully exploited.

The decompression of a patch reconstructs the quad-tree level corresponding to the level of the image pyramid which has been selected for rendering. As the result of the decompression the reconstructed quad-tree tiles will be stored in a vertex buffer. The vertex buffer contains four floating point values per reconstructed point or six if normals are also decompressed. These values are the height value h, two coordinates u and v specifying the location of the point in the primitive's domain, as well as a value b that is zero if the point corresponds to a position that was masked out by the occupancy bitmap. In case of decompressed normals there are two additional values ϕ and θ. Since each quad-tree node specifies an index of a single code-vector, a node decompresses into exactly x^2 points. Using (u, v) the height and normal values of the points will be transformed into world-coordinates with respect to the shape primitive in a vertex shader during rendering. Points with $b = 0$ are discarded in a geometry shader.

Thus, for reconstruction of a level P_j, the arrays of the quad-tree for level j are uploaded into textures on the GPU. The reconstructed nodes of the previous level P_{j+1} are copied into textures as well. If the number of nodes in the quad-tree on level j is k, than kx^2 points have to be reconstructed, since every node encodes a x^2 image tile. We use the transform feedback OpenGL extension, render kx^2 points and perform the decompression of each point in a vertex shader. The transform feedback stores the result of the vertex shader directly into a vertex buffer which can then directly be used for rendering without any prior copying.

The decompression vertex shader reads the respective node information from the quad-tree array. Note that to achieve maximal concurrency the node information is actually read many times - once for each point. Due to caching of the data this causes virtually no overhead however. All that has to be done for reconstruction is to add the code-vector entry, which is read from a codebook texture, to the respective interpolated parent value. The interpolation is handled automatically in the texture unit. Additionally, the point's coordinates are deferred from the parent coordinates. Note that the interpolation of parent values is restricted to the values belonging to the parent node by adjusting the texture coordinates accordingly. This causes a slightly decreased compression performance but offers the advantage that no neighboring quad-tree nodes have to be considered in the decompression. Thus, computations as well as data structures are significantly simplified.

3.6 Rendering

In principle, during rendering, the decompressed height values only have to be transformed into 3D coordinates in a vertex shader and can then be rastered as simple point primitives. However we want to incorporate level-of-detail control for better performance. Also hole-free renderings of close-up views are desired.

3.6.1 Level-of-detail

With our system, level-of-detail control can be achieved fairly simply: For each patch the distance d of the bounding box to the viewer is obtained. Since the sampling resolution of the patch is known, this distance can be used to select the level-of-detail as follows:

$$l_{LOD} = -\log_2\left(\frac{\sqrt{2}nr_{patch}}{dp}\right), \tag{3.9}$$

where r_{patch} is the resolution of the patch, n is the distance to the near plane and p is the side length of a pixel in world coordinates. This choice guarantees a hole-free rendering of objects as long as $l_{LOD} \geq 0$. For patches with $l_{LOD} < 0$ we use a hierarchy of coarser framebuffers to obtain hole-free renderings. The levels of this framebuffer hierarchy are merged into a single image for each frame.

3.6.2 Hole-free rendering

To achieve a hole-free point rendering, splatting approaches such as those proposed in [ZPvBG01] or [BK03a] usually are a first choice. Splatting however requires a normal for each surface element so that it cannot be directly applied if normals have to be estimated in image-space. Also splatting requires the geometry to be rendered in two passes, which causes significant overhead for large models. The pyramid of framebuffer images that we employ instead requires only a single geometry rendering pass followed by a few very fast image-based passes. This is similar in concept to the point sample rendering of Grossman and Dally [GD98]. However, we use a different GPU supported depth buffer technique and propose a new splat-based merging strategy to combine framebuffer levels.

On the GPU we use a single off-screen framebuffer that is large enough to contain the images of all pyramid levels. To render into a specific pyramid level only the viewport needs to be adjusted accordingly. This way each pyramid level has its own, separate depth buffer and therefore may contain parts that will not be visible in the final image. Instead of color values, we store the points' positions and normals in the framebuffer. Additionally a radius is stored for each point. The radius r_{frag} is derived from the resolution with which the respective patch has been rendered, i.e. $r_{frag} = 2^{\max(l_{LOD},0)} r_{patch}$. This way each pixel encodes the parameters of a circular splat.

To merge the pyramid levels into a single image, the framebuffers are processed from coarse to fine. To this end the coarse level and the next finer level are bound as textures and rendered into a new texture with the same dimensions as the finer level to yield the combined image. A screen aligned quad is used to start a fragment shader for each pixel of the combined image.

The fragment shader intersects the splats encoded in the pixels of a small neighborhood in the coarse image with the ray corresponding to the combined image's pixel. The location of the intersection is used to evaluate an object-space kernel for each splat which determines its influence for the pixel at hand. We use a simple Gaussian g_σ kernel with $\sigma = \frac{1}{2} r_{frag}$. The blended contribution b from the coarse level can thus be obtained as

$$b = \frac{1}{\sum_i g_\sigma(q_i - p_i)} \sum_i g_\sigma(q_i - p_i) a_i, \tag{3.10}$$

where p_i is the position of the splat and q_i is the splat-ray intersection and a denotes any of the attributes position, normal or radius.

Since it is not guaranteed that splats from the coarse image always occlude those in the finer image, we perform a depth test before writing the blended coarse image to the result image. Should the depth test fail, the splat from the fine image is written.

The blending of pyramid levels proceeds until the finest resolution has been reached. Then, in a last step, deferred shading is applied to generate the final on-screen rendering.

3.6.3 Normal estimation

If the point-cloud was compressed without normal information, normals have to be estimated on-the-fly during rendering. To this end, we first render the points in the framebuffer pyramid as described above and then compute the normals in a second pass [KK05], similarly to deferred shading. This has to be done before the pyramid levels are merged in order to obtain valid splats.

In the normal estimation pass, the point positions in $5x5$ blocks of pixels are used to estimate normals. We use weighted least-squares to fit a plane to the points via PCA of the covariance matrix [HDD+92]. Again we use the object space Gaussian kernel g_σ to determine the influence of the neighbor points, where σ is computed using r_{frag} of the center pixel as described above. Since the weights are obtained in object space, no notable smoothing occurs across edges in the image.

model	N	R_{max}	T_D	T_{VQ}	bpp	disk1 bpp	disk2 bpp
St. Matthew	186,810,938	0.1mm	1:28	0:13 (1:07)	1.95 (3.15)	1.57 (2.72)	1.1 (2.07)
Atlas	255,035,497	0.1mm	2:13	0:05 (1:52)	1.41 (3.0)	1.15 (2.49)	0.77 (2.11)
David	28,184,526	0.1mm	0:11	0:05 (0:24)	1.93 (4.01)	1.52 (3.2)	1.21 (2.89)
Ephesos	23,073,902	1mm	0:16	0:05	2.37	2.16	1.67
Industrial	23,207,348	5mm	0:21	0:02	1.75	1.47	1.1

Table 3.2: Compression statistics for various models. T_D gives the time for decomposition in hours and minutes. T_{VQ} is the time for vector quantization. All timings were obtained on an Intel Core 2 Duo with 2GB Ram. Bpp are measured with respect to original points. Disk1 bpp uses on-disk compression using simple adaptive arithmetic coding while disk2 bpp lists results for breadth-first serialized quad-trees. Timings and bitrates in parentheses are for points and normals. For each model six levels-of-detail were used. For Ephesos and Industrial no normals were compressed due to their low quality.

The normal estimation is executed once for all pyramid levels by drawing a screen-aligned quad over the whole off-screen framebuffer. After that the merging of levels proceeds just as described above.

A problem may occur during the normal estimation if areas of the object are viewed in a grazing angle. Then it can happen that neighboring pixels contain only points that are so distant to the center pixel's point that their weight becomes zero due to numerical reasons. In such a case the normal estimation can produce arbitrary results. While this is a principle drawback of image based normal estimation, in our case we can greatly alleviate the problem by incorporating the additional information available in the form of point normals generated from the underlying primitive shape. This normal can be used to appraise the angle under which the point is viewed and the point's radius can be enlarged accordingly. Thus, we set

$$r_{frag} = \frac{1}{< n, v >} 2^{\max(l_{LOD}, 0)} r_{patch}, \tag{3.11}$$

where n is the shape normal of the point and v is the viewing direction. Since σ is directly correlated with r_{frag} the kernel width is adjusted implicitly as well. Note that the enlarged r_{frag} also increases the size of the splats used during merging of pyramid levels which fills spurious gaps between splats that can occur for steep viewing angles.

3.7 Results

In order to evaluate our system we conducted several experiments. Table 3.2 lists the bitrates achieved by our method for various models. The bitrates of our

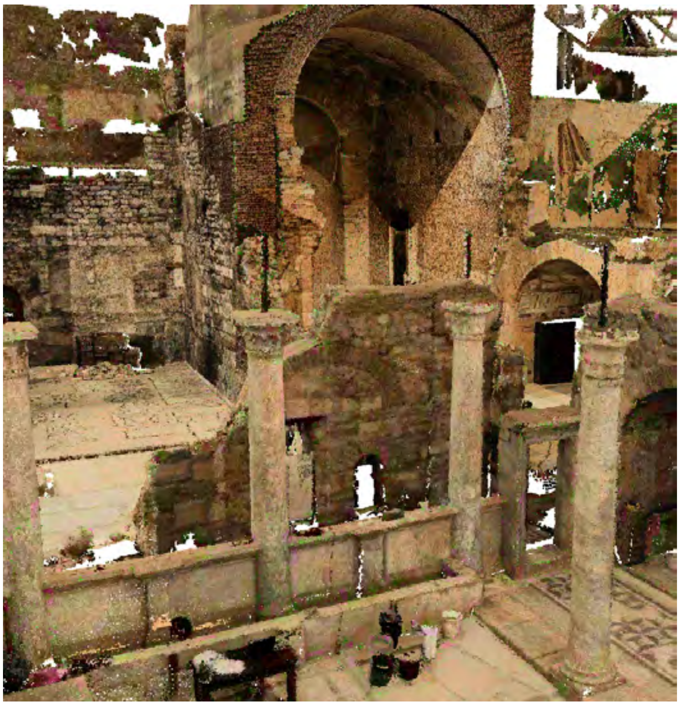


Figure 3.5: The Ephesos point-cloud with colors after compression.

model	N	bpp
Santa	75,781	1.36
MaleWB	148,138	2.07
FemaleWB	121,723	1.84
Ephesos	23,073,902	2.3

Table 3.3: Compression statistics for colors on various models. Bpp gives the bits required for colors only, i.e. without heights or normals.

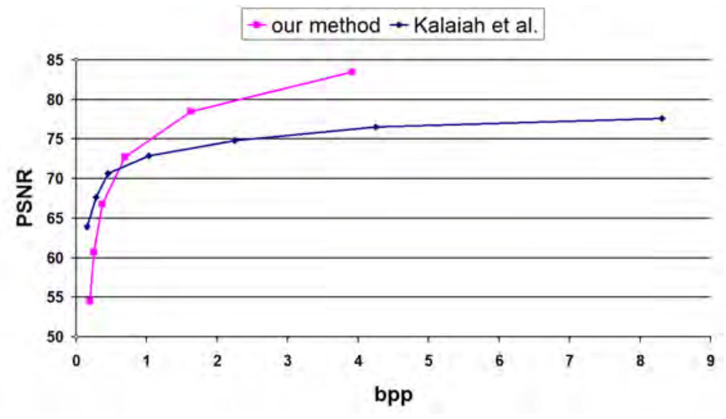


Figure 3.6: The PSNR of our method compared to that of Kalaiah et al. [KV05] for the David statue.

method are of the same order as results reported on smaller models by previous sequential coders. Where applicable we compressed normals with $R_{max} = 1°$. In all cases we used an image tile side length of $x = 4$. Note that our method performs better on larger models as the codebook costs are better amortized. For all models, the ϵ parameter of the shape detection was set to equal three times the desired RMS. The parameter α was set to 30 degrees, so that parameterizable patches were found in all cases. Shape parameters and bounding boxes require between 0.2-0.3bpp and occupancy bitmaps about 0.4bpp. Only for the extremely irregularly sampled long-range scans of Ephesos and the industrial compound (see Fig. 3.9), the bitmaps take up roughly 1bpp due to the many holes and complex boundaries in the data. Due to several scanning artifacts in these scans the normals computed in a preprocess are of low quality so that they do not provide significant improvements over our screen-space estimation scheme. Thus we chose not to compress them. Also note that for this data it is extremely valuable that our method is able to adjust the sampling density locally for each patch, which is impossible to achieve with a global grid as employed by [KSW05]. Table 3.3 lists the bpp required for additional compression of colors. For the compression a PSNR of 34 was used for the Y channel and a PSNR of 32 for the UV components. For a result see Fig. 3.5.

In Fig. 3.6 a comparison of our method with the approach of Kalaiah et al. [KV05] is given. The error was measured as described in their work and the peak signal is given by the bounding box diagonal. While our method performs slightly worse for low bitrates below 0.7bpp, a high PSNR above 75 is achieved with far fewer bits. For a PSNR of 78 our system requires less than 50% bits than their method.

In order to asses the effect of the extended set of primitive shapes, we com-

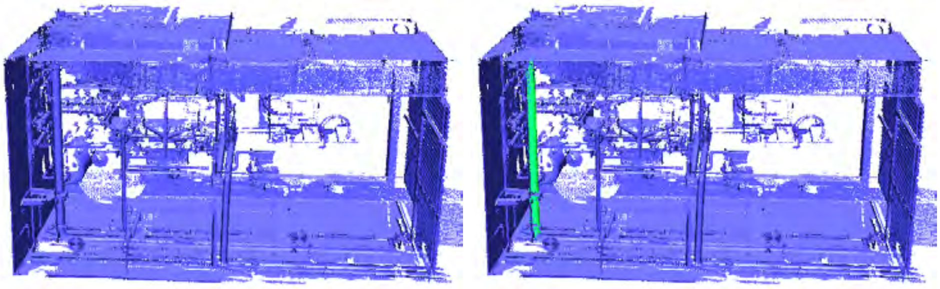


Figure 3.7: Some simple interaction trivially supported in our system. A pipe is highlighted by clicking on it.

pared results of our system with all primitive shape types activated to results for which only planes were allowed. Obviously the benefit of the extended set of primitives depends on the type of geometry. On the one hand, for objects in which planar areas dominate or in which neither planes nor other shapes are actually present, none, or only a very small, gain can be achieved with the extended set of primitives (e.g. for the Michelangelo statues). On the other hand, for objects such as the oil pump (see Fig. 3.2) or the industrial compounds in Fig. 3.7 and 3.9 the extended set of shapes is a distinct advantage, improving the bitrate about 12%. Since planes are included in the extended set as well, we never observed an increased bitrate when all shapes were activated.

Fig. 3.8 shows two close-up images generated with our framebuffer pyramid. Holes are smoothly filled while detail is retained. In the image on the left normals were estimated in image-space before upsampling of the framebuffer levels. At such a scale small artifacts in the estimated normals may become visible. Fig. 3.10 gives two images to illustrate the performance of the normal estimation at another distance from the viewer. It can be seen that the estimated normals generally introduce a certain amount of smoothing. For larger distances this smoothing is almost equivalent to screen-space anti-aliasing but for closer views some of the detail may get lost due to the limited screen resolution. Note that detail becomes visible when the screen resolution roughly matches, or is finer than, the model resolution (see Fig. 3.8). The level-of-detail rendering ensures that this is mostly the case. Estimating normals in screen-space takes about 6ms per frame. In certain scenes where primitive shapes are predominant, e.g. the industrial compound shown in Fig. 3.9, rendering of shape normals alone already suffices to create a realistic impression.

On a GeForce 8800 GTX we currently achieve framerates between 5 to 10 fps for large models, such as Atlas or St. Matthew. We do not apply any culling techniques besides frustum culling. For some models, back-face culling would result in considerable speed-up, but since many point-clouds are non-manifold (e.g. Fig. 3.7 and 3.9) back-face culling is not appropriate in general. We do plan on integrating occlusion-culling in the future. Decompression speed varies between 5-10 million points per second, depending on the levels that are decoded. Coarser levels are slower because of the render call overhead. Compared to our parallelized CPU implementation this is a speedup of about a factor of 10 (measured on an Intel Core 2 Duo).

Fig. 3.7 shows a small example of the interaction that is supported by our compression format. Parts corresponding to shapes can easily be suppressed or highlighted for visualization purposes. Moving or duplicating such parts would be possible as well, but we have not implemented this form of interaction yet.

3.8 Conclusion

We have presented a progressive compression scheme for point-clouds that aims for fast parallel decompression while achieving lower bitrates than other state-of-the-art compression algorithms which aim at direct rendering. This is facilitated by the close geometric approximation as well as the simple local parameterizations provided by the detected shape primitives. With this help, simple image-based compression techniques can be employed and we have demonstrated that, using Vector Quantization, the decompression can be executed well on today's GPUs, enabling inspection of the compressed geometry in interactive rendering. In order to support efficient parallel processing several compromises in the layout of the compressed format were made. For instance in the quad-tree hierarchy every node redundantly stores a pointer to its parent so that nodes can be processed independently. We also showed that with the use of arithmetic coding further compression gains can be achieved for on-disk storage. However, streams generated with this technique cannot straightforwardly be decompressed in parallel, but have to be transformed to the parallelly decodeable format during loading. But even in the GPU decompressable version, our current system's compression rate on large models is of the same order than that of previous sequential coders. To allow the quick compression of large models several strategies for acceleration of the vector quantization have been proposed and their effectiveness was demonstrated.

3.8.1 Limitations and future work

The level-of-detail obtained with our approach is not suitable for very far objects, i.e. objects filling only a couple of pixels on the screen. This is due to the fact that no filtering takes place across shape borders and the number of patches is not reduced for distant views. Our approach could be extended by adding volumetric

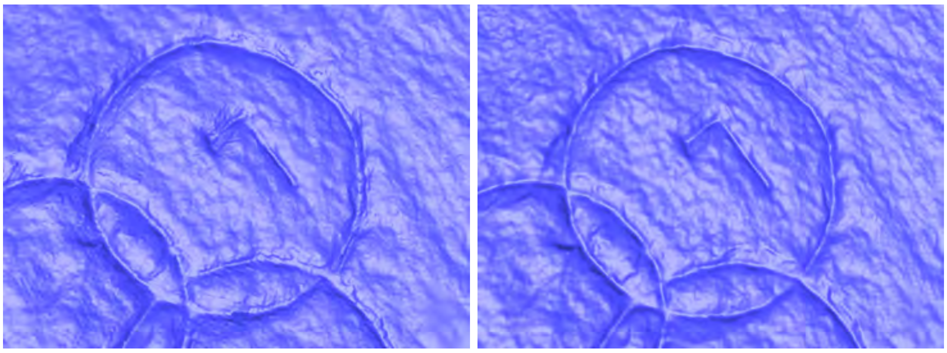


Figure 3.8: Close-up of fine detail on Michelangelo's Atlas. Hole free rendering is achieved with our framebuffer pyramid. On the left decompressed normals are used. On the right normals have been estimated in screen space.

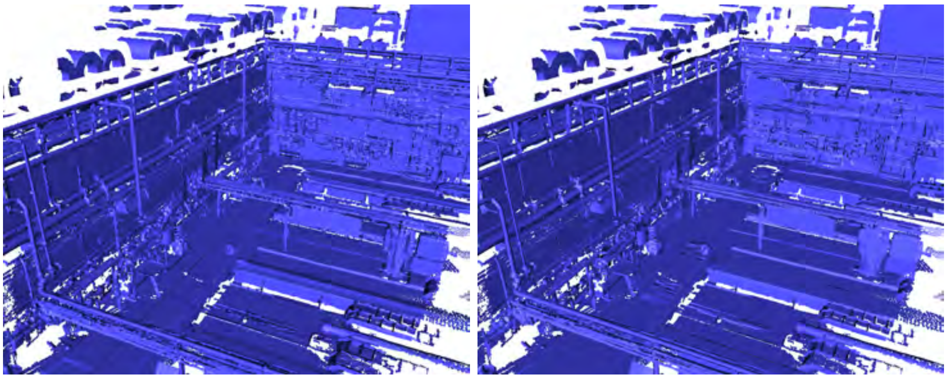


Figure 3.9: The image on the left has been rendered with normals estimated in screen-space. On the right only shape normals are shown.

hierarchies for very coarse views, resulting in a structure reminiscent of e.g. VS-Trees [BHGS06]. Exploring these possibilities is a main avenue of future work. We also plan to accelerate the rendering by incorporating occlusion culling and reducing the render call overhead for coarse levels by combining different patches in a hierarchy.

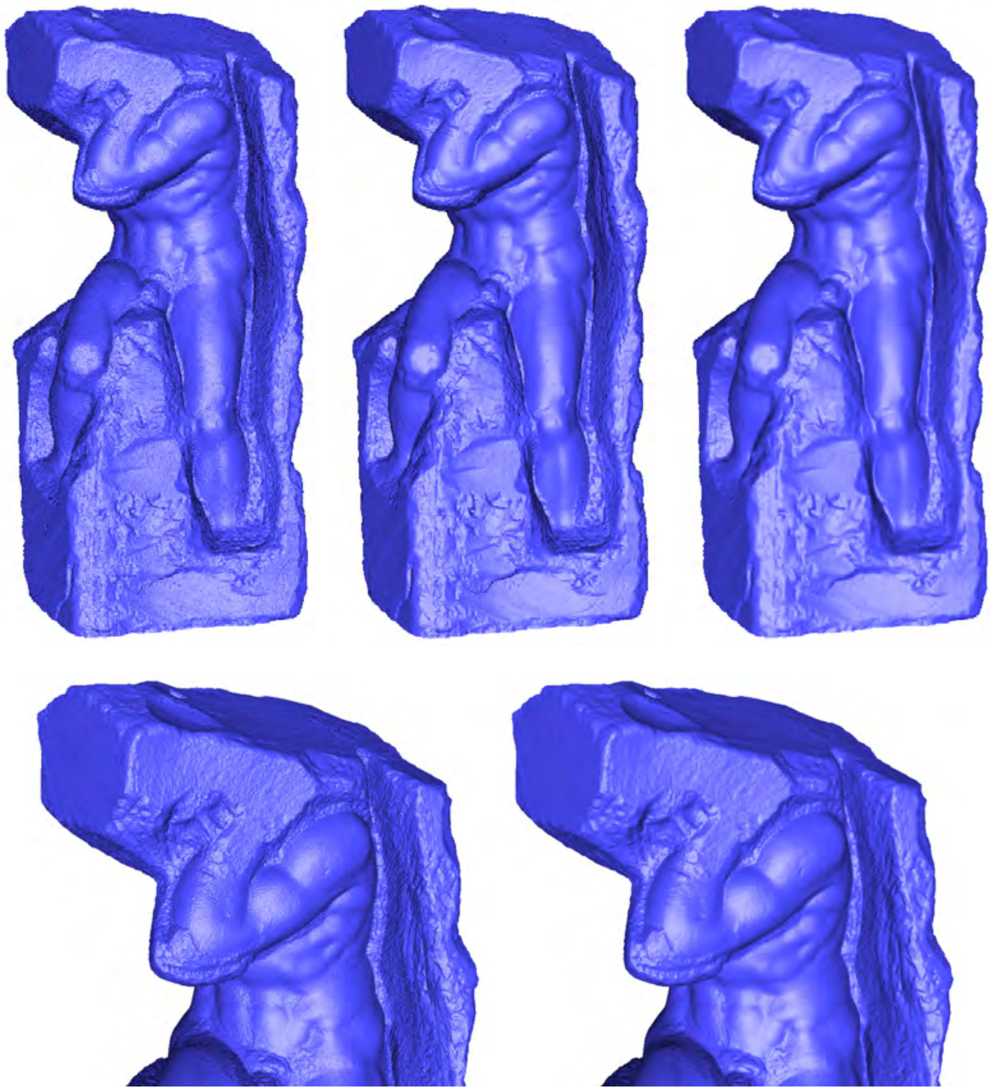


Figure 3.10: The behavior of the normal estimation for different distances from the viewer. In the top row the Atlas model has been rendered with compressed normals on the left. In the middle screen-space aliasing was achieved by splatting the points. On the right normals have been estimated. The bottom row shows a zoomed in view. On the left decompressed normals were used and on the right estimated normals are shown.

CHAPTER 4

RECOGNITION

4.1 Introduction

While the previous chapter has demonstrated the effective use of shape primitives for compression and visualization of large and detailed point-clouds, interaction capabilities on a semantic level are still very limited. Even tasks as basic as selecting all windows in a scan of a house require a disproportional amount of user interaction. The extraction of such semantic elements from 3D point clouds is an important topic for a wide field of applications, including architecture, cultural heritage and city model reconstruction. For instance, semantic enrichment of 3D building models is of large interest to architects: Automatic retrieval and classification of building units may allow to automatically determine the style of a building. Moreover, automatic identification of building units supports comfortable cut-and-paste editing. Further applications can be envisioned in the engineering context, e.g. automatic inventory of industrial plants, robotics or traditional reverse engineering processes.

This chapter presents a method that exploits both the concise description of the input point-cloud as well as the classification provided by detected shape primitives in order to recognize predefined surface structures. The two important building blocks of the algorithm are:

Decomposition By decomposing the point-cloud using shape primitives, we obtain an abstraction of the data that eliminates much of the redundancy. A *topology graph* captures the neighborhood relations between the primitives in a concise manner.

Constrained subgraph matching Semantic elements such as e.g. windows, roofs or columns can be described by characteristic configurations of primitive shapes. These configurations are captured in *query graphs*. Semantic entities can then be detected in the topology graph via constrained subgraph matching.

The system allows even unexperienced users to quickly formulate complex configurations, since primitive shapes are easily graspable and combinable and are able to describe very different kinds of semantic entities on a geometric level. Moreover the method supports optional and repetitive components in the query graph so that generalized classes of semantic elements can easily be captured in a concise and intuitive manner.

Our method was tested on a large amount of data including point-clouds from different kinds of sensors like *LIDAR (light detection and ranging)* and stereo reconstruction. We also applied our method to 3D CAD modeled buildings that include interior structures. Due to the use of the detected primitives the approach is robust against noise, clutter, registration errors or miscalibrations which are frequently encountered in 3D laser scans.

4.2 Related work

Our shape matching technique is related to many works in the larger context of (partial) matching, classification and retrieval of 3D shapes. Various approaches to these challenges have been developed in the past. Here we concentrate on methods for partial matching and retrieval of 3D objects in larger 3D scenes. For a more detailed introduction to 3D matching and shape retrieval we refer to [TV04].

4.2.1 3D city reconstruction

Since in this chapter we mainly deal with data from the architectural domain, automatic reconstruction of 3D buildings and city models from aerial LIDAR data, a special case of the above formulated tasks, is especially relevant to our approach. In this setting detection is restricted to simple shapes of which most can be described by configurations of planes. Automatic building reconstruction has been studied extensively in the photogrammetry community. Most of the developed semi-automatical or manual approaches rely on user interaction or on semantic knowledge that is not contained in the 3D point-cloud. For Example, Vosselman [Vos02] integrates LIDAR data with a 2D Geographic Information System (GIS) database and aerial photographs. The GIS delivers ground plan information about buildings in the point cloud. With this information at hand, the points belonging to a single edifice can be extracted and parametric building models can be fitted.

A fully automatic approach that is only relying on the information contained in the point-cloud is presented by Verma et al. [VKH06]. They use a region growing approach to detect planes in the point data. Roof-topology graphs are defined to describe configurations of planes for some simple building forms shaped like I, L and U. These configurations are sought in the set of the detected planes. In a second step, the detected simple buildings are extended to more complicated forms according to the plane configurations in the point-cloud. Compared to our approach, there are two differences: First, as only planes are detected in the LIDAR data, the approach is restricted to those shapes that can be decomposed into planar patches. Second, the method does not use any node or graph constraints during the subgraph search and is therefore susceptible to misclassifications in more general settings.

4.2.2 Graph-based matching

The retrieval and matching of 3D objects is of particular interest to the computer graphics community. The developed methods often rely on triangle meshes or parametric representations of the objects. Among the abundance of proposed approaches, several graph-based shape retrieval methods rely on the extraction of certain geometric components and use a graph to capture the relations between these components.

Model graph-based approaches are mainly used for geometry such as is generated in CAD applications. Model graphs describe solid objects in terms of connectivity of freeform surfaces (Boundary Representation) or as a set of geometric primitives that are connected by Boolean operations (Constructive Solid Geometry). Local clique matching [EM03] [EMA03] or comparison of graph spectra [MPSR01] are used to globally determine the similarity of the graphs, i.e. no partial matching is supported. In [ZTS02], VRML objects are segmented according to different decomposition techniques. The resulting patches are assigned basic shapes like planes and spheres. An attributed decomposition graph is built containing the determined shapes as nodes. Neighboring shapes are connected by edges. The similarity between two objects is computed by matching the associated decomposition graphs using error-correcting subgraph isomorphism (a recognition paradigm that was first introduced by Nevatia and Binford [NB77]). However they cannot handle user defined query graphs nor do they consider optional or repetitive components.

Reeb graph-based methods rely on a function that is computed on the model surface. The surface is divided into segments corresponding to intervals of this function. A skeleton graph, in which the resulting segments are represented as nodes, is built. Reeb graph-based methods are mainly used for matching of articulated objects [HSKK01][TL07]. In [PSBM07], a robust method for fast Reeb graph computation is proposed that even allows for the use of non-manifold meshes. However, Reeb graph methods do not allow a user to easily describe sought components in an intuitive manner but rely on specific example geometry which might not always be available.

Skeleton graphs of 3D shapes can be computed using topological skinning of voxel representations [BNdB99], medial axis transform, ridge point tracking [CSM05] or deformable model-based reconstruction [SLSK07]. Matching of two shapes is done by comparing the associated skeleton graphs using greedy bipartite graph matching [SSGD03] or by detecting subgraph isomorphisms using decision trees [LJI^{+}03].

4.2.3 Matching with local features

In general, these approaches consist of two steps. In the first step, local features and shape descriptors are computed. In the second step, the similarity between two objects is computed according to the similarity of the associated descriptors. Compared to our approach, the employed local features and descriptors are unintuitive and do not allow simple and generalized descriptions of sought entities but focus on automatically deriving suitable descriptions for given geometry in order to retrieve similar surface parts.

In [GCO06], local surface descriptors for triangle meshes are introduced. A voting scheme is used to determine self-similarities, alignments and partial matchings in larger scenes. Partial matching of two shapes is addressed in [MGGP06] where shape signatures based on spin-images are computed from surface patches. Based on these signatures, a fast probabilistic search framework is used to estimate partial similarity between two shapes. In [MGP06] Mitra et al. use local features to perform a voting in transformation space in order to detect approximate symmetries on 3D objects. Mean-shift clustering in transformation space is used to identify the most salient transformations which are likely to correspond to symmetric surface parts.

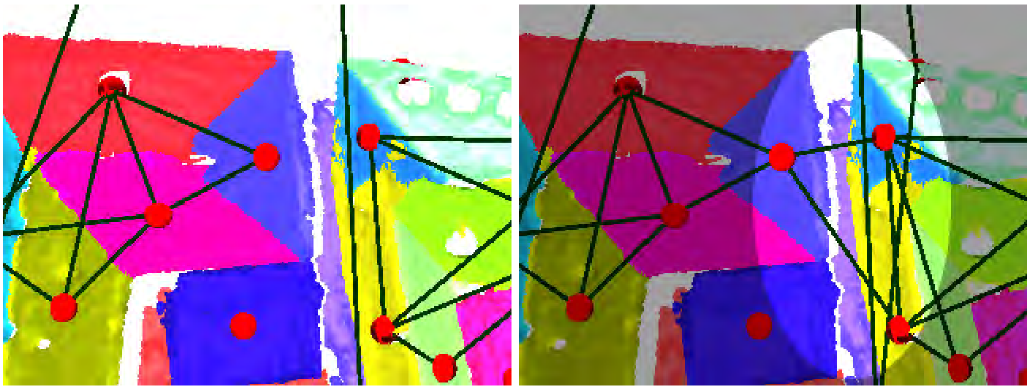
In [SF06], [FS06b] and [SF07], a distinctiveness measure for local Spherical Harmonics-based descriptors is introduced. Distinctiveness is determined with respect to a pre-classified database of 3D objects. Those features providing a certain amount of distinctiveness are used to establish partial matchings between two 3D objects.

A hybrid between graph-based and feature based approaches was presented by Berner et al. [BBW^{+}08]. They detect local features based on a slippage analysis and connect nearby features in a graph. Symmetries in the model are then discovered by a randomized search for repetitive subgraphs. Bokeloh et al. [BBW^{+}09] later augmented the approach by using crease-line features instead.

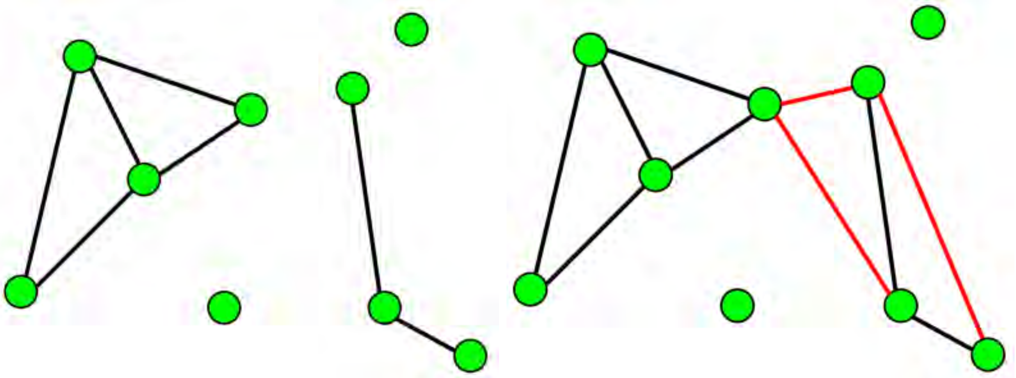
4.3 Overview

The algorithm consists of two main steps: Firstly a shape based representation of the data is derived by detecting primitive shapes in the unstructured point data and constructing a *topology graph* that captures the neighborhood relations between the different shapes. Then, in the second step, this topology graph is searched for characteristic subgraphs corresponding to sought elements, as they have been defined by the user.

The user is able to quickly define and retrieve new entities with geometric constraints in an interactive framework. Moreover the detected subgraphs may contain optional or repetitive components, which further simplifies the definition of new entities for the user. This way our method is very flexible and easily extensible, which renders it suitable in a broad range of applications.



(a) Small cell size (b) Large cell size



(c) Topology graph for small cell size (d) Topology graph for large cell size

Figure 4.1: Two houses viewed from above that are separated by a narrow alley. Primitive shapes have been detected and are depicted in random colors. a) The topology graph was built with a cell width of 50cm b) The cell width for the construction of the topology graph was set to 2m. Note that the roofs have been connected across the narrow alleyway. In c) and d) we show the resulting topology graphs. In d), the additional edges resulting from large cells are shown in red.

4.4 Topology Graph

The topology graph $G(\Psi, E)$ describes the neighborhood relations between the primitive shapes detected in the point-cloud data. For each primitive ψ_i a vertex is inserted into the graph, i.e. $\Psi = \psi_1 \ldots \psi_N$. Two shapes are connected with an edge if their supports are neighboring in space, i.e. the two vertices ψ_i and ψ_j are joined by an edge $e = (\psi_i, \psi_j)$ if

$$\exists p \in \mathcal{P}_{\psi_i}, q \in \mathcal{P}_{\psi_j} : \|p - q\| < t \tag{4.1}$$

holds, where S_{ψ_i} is defined as in Eq. (2.12) and t is a user specified distance threshold. Please note that computing the distance between the shapes directly and ignoring the support would result in many edges that have no counterpart in the data, since the shape primitives have indefinite extent.

Thus, to find the graph edges, the spatial proximity between the support of all detected shapes has to be determined. To this end we employ an axis aligned 3D grid. In a first step all points are sorted into the grid. Then for all grid cells that contain points belonging to different shapes, edges connecting the corresponding graph vertices are added to the graph, i.e. for each pair of shapes in the cell an edge is created. In order to avoid discretization dependencies due to the location of the grid cells, we use eight shifted and overlapping grids. Cells are stored in a hash table, so that memory is only allocated for occupied cells.

The width of the grid cells defines how far apart shapes are allowed to be in order to still get connected in the topology graph. Given the distance threshold t, the width of the cells is set to t and the shifted versions of the grid are created with an offset of $\frac{1}{2}t$ along the respective axes. Of course this means that shapes can get connected in some cases even though the distance between their support is in fact only less than $\sqrt{3}t$. It would be possible to eliminate these cases by checking the distance between the points in each cell, but we found this additional overhead unnecessary in practice, as the few errors in the less restricted topology graph did not influence the performance of our algorithm. In Figure 4.1 resulting graphs are depicted for different cell widths.

Once the graph is complete the first step of our method is finished and a shape based representation of the point-cloud data has been derived. It can now be used to efficiently detect more complex configurations of primitive shapes that correspond to semantic entities in the data.

4.5 Shape Matching

In order to achieve an automatic matching between semantic entities and point-clouds, we have to find a common language for them. As we abstracted the point data to obtain a higher level description, we have to concretize the representation of feature elements in terms of primitive shape configurations.

4.5.1 Query graph

To this end we define a *query graph* for an element as a graph that captures its characteristic shape configuration. Basically a query graph is a topology graph with the difference that it does not stem from a point-cloud, but from knowledge about the shape of an element which is introduced by the user. The recognition of an element in the data then corresponds to a matching of the query graph to a subgraph of the topology graph. Even though subgraph matching is a NP-complete problem [Coo71], in our applications the query graphs will be small, i.e. usually less than twenty vertices, matching is highly constrained so that a simple brute-force implementation of subgraph matching performs well in such a setting.

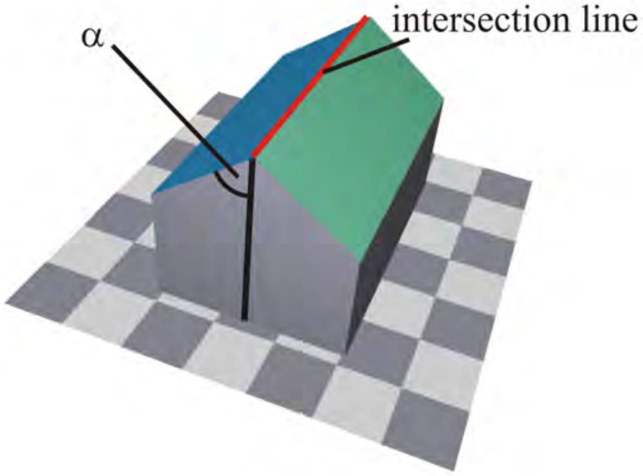


Figure 4.2: Illustration of the constraints that can be used to detect saddleback roofs: The angle α is constraint to be less than 90 degrees and similar for both planes. The intersection line is required to run parallel to the ground.

However, a representation solely based on topology is not sufficient to discriminate between many different feature elements. For example in the simple case of a saddleback roof (see Figure 4.2), the query graph consists of two vertices corresponding to planes connected by an edge. If such a graph is searched in a topology graph numerous false matches can be expected, as such a simple configuration occurs frequently. To make the detection more reliable, the user can add constraints to the query graph. For instance, in the case of the saddleback roof we require the two planes to exceed a certain size, to be of similar size, and to intersect in a line that is parallel to the ground.

If we take a closer look at these geometric constraints, we find that they can be divided into classes that access different kinds of information. There are *node constraints* which only restrict the primitive shape associated with a node (e.g. type, size or orientation). There are *edge constraints* which restrict the relation between two incident shapes (e.g. angle between two planes). Any constraint not fitting into one of the first two classes belongs to the class of *graph constraints*, because it relies on the topology to be checked (e.g. sums of sizes, parallelism of disconnected planes).

Thus, when modeling a query graph the user specifies the sought shape configuration on a topological level by insertion of shape nodes and connecting edges. Geometric relations, however, are attached to these graph elements in the form of constraints which are formulated in a simple scripting language. The scripts have access to all parameters of the supported primitives as well as to the set of assigned points for each shape. Moreover, predefined functions for computation of intersections, test for parallelism or orthogonality etc. exist.

4.5.2 Constrained subgraph matching

The outline of the recursive constrained query graph matching is illustrated in Algorithm 3. To simplify the discussion of the procedure, no explicit statements are given for neither the maintenance of a data structure storing the matching, nor keeping track of visited nodes. However, these actions are assumed to take place implicitly and it should be noted that they are mandatory for any correct implementation of the method. In the following the different parts of the algorithm will be described in detail:

In lines 2-8 the outer matching function is given, which searches for a suitable node in the topology graph, where the matching can be started. This outer matching function has to be started repeatedly to retrieve all possible matches.

Matching a node In lines 10-18 the function for matching a node is sketched. First a check is made to see if all edges of the node have already been matched and if this is the case, the matching of the node has been successful (lines 11-14). Otherwise a yet unmatched edge of the node is chosen and matched to an edge of the topology graph by calling the MatchEdge function. If the edge was matched successfully, MatchNode is called recursively on the same node, in order to match any remaining edges of the node (lines 15-18).

Algorithm 3 Recursive Constrained Subgraph Matching

```
Input A topology graph $T = (V_T, E_T)$ and a query graph $Q = (V_Q, E_Q)$
Function MatchSubgraph(Q, T)
$v_Q \leftarrow$ StartNode($V_Q$)
for all $v_i \in V_T$ do
    if CheckNodeConstraint($v_Q$, $v_i$) then
        if MatchNode($v_Q$, $v_i$) then
            return true
        end if
    end if
end for
return false

Function MatchNode($v_Q, v_T$)
if $v_Q$ has no unmatched edge then
    if all nodes in $V_Q$ are matched then
        return CheckGraphConstraint()
    end if
    return true
end if
$e_Q \leftarrow$ first unmatched edge of $v_Q$
if MatchEdge($e_Q, v_T$) then
    return MatchNode($v_Q, v_T$)
end if
return false

Function MatchEdge($e_Q, v_T$)
for all unmatched outgoing edges $e_i$ of $v_T$ do
    if CheckNodeConstraint(dest($e_Q$), dest($e_i$)) then
        if CheckEdgeConstraint($e_Q$, $e_i$) then
            if MatchNode(dest($e_Q$), dest($e_i$)) then
                return true
            end if
        end if
    end if
end for
return false
```

Matching an edge In lines 20-26 the MatchEdge function is outlined. It checks if any of the unmatched edges of the given topology graph's node can be matched to the given query graph edge (line 21-25). This is the case if the end-nodes of the edges can be matched successfully - which is tested via a call to MatchNode (line 24).

Checking constraints Constraints are always verified just before a match is established (lines 5, 13, 22, 23). In line 13 the graph constraints are checked as soon as all nodes of the query graph have been matched. If this test fails, the matching will backtrack and continue the search. As can be seen, in contrast to graph constraints, both, node and edge constraints, have the advantage that they can be checked early on during the subgraph matching procedure, as they do not rely on other parts of the graph. This is an important performance factor since this way many of the topologically correct matches can be quickly discarded, without causing extensive backtracking.

In order to avoid the need for graph constraints when using asymmetric edge constraints (e.g. sphere A has to be larger than sphere B) we also support directed edges as carriers of a constraint.

4.5.3 First results

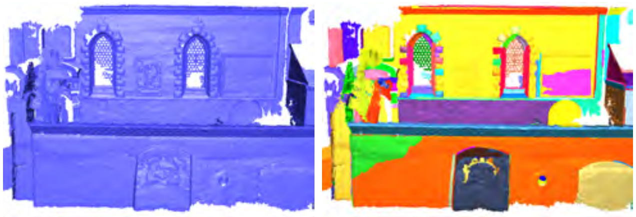
At this point the methods presented so far are powerful enough to recognize large classes of semantic entities and we will first give a couple of examples illustrating the possibilities before presenting further extensions to the basic matching framework:

In Figure 4.3 a query graph was designed to detect Gothic windows in a scan of a medieval chapel. The windows were modeled as two spheres for the arches and two planes for the sides of the window. The spheres were constrained to have roughly equal radius and the planes to be tangential to the spheres.

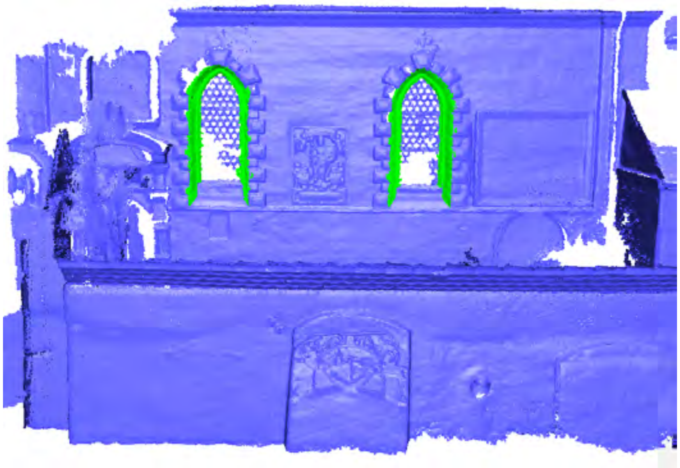
To find the columns of the choir screen in Figure 4.4 they were modeled as a cylinder connected to two tori at both ends. The cylinder and the tori were constrained to posses the same axis of rotation (with a small tolerance of 5 degrees).

4.5.4 Query Graph Extensions

Although the given definition of a query graph already covers many shape configurations, there remain cases which it is still insufficient for. In the following we discuss some of these cases and demonstrate how they are overcome by extensions to the query graph model:



(a) Points (b) Shapes

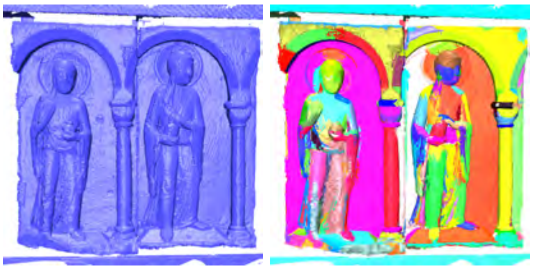


(c) Windows

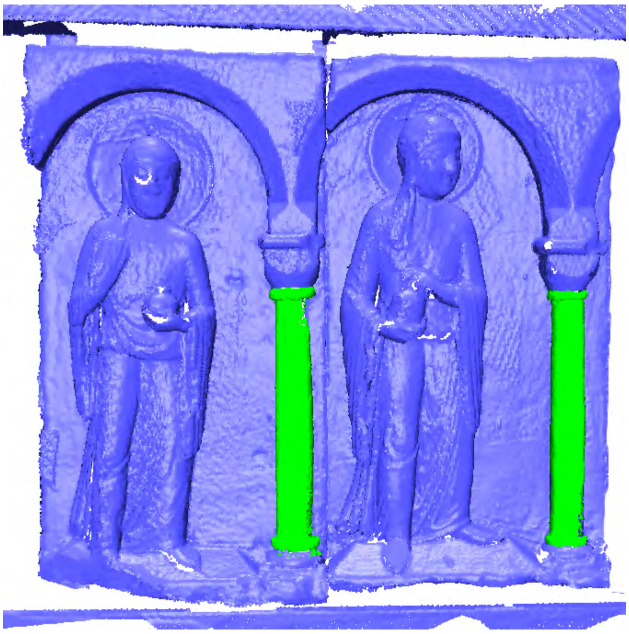
Figure 4.3: A scan of a medieval chapel with Gothic windows containing 4.2M points. The windows were detected by matching the query graph with subgraphs of the topology graph. In a) the original point-cloud is depicted. b) shows the support of the detected shape primitives in random colors. In c) the detected columns are highlighted in green.

Context nodes Certain features benefit from a *context object* to distinguish them from other structures. For instance, a balcony needs a wall as context. Therefore we need to model the context in the query graph, but without declaring it an integral part of the balcony. This is achieved by tagging these query graph nodes as context nodes, so that after searching they can be removed from the match. In Figure 4.5 a) we give an example for this concept. There the roof planes have been modeled as the context shapes of the dormers, see Figure 4.5 b).

Optional nodes A limitation of the way we model queries so far is that we are not able to specify variants of an entity without duplicating the original query graph. For instance L-shaped roofs like the one shown in Figure 4.6 may occur in four variants: not hipped, hipped on either end or hipped on both ends. Thus



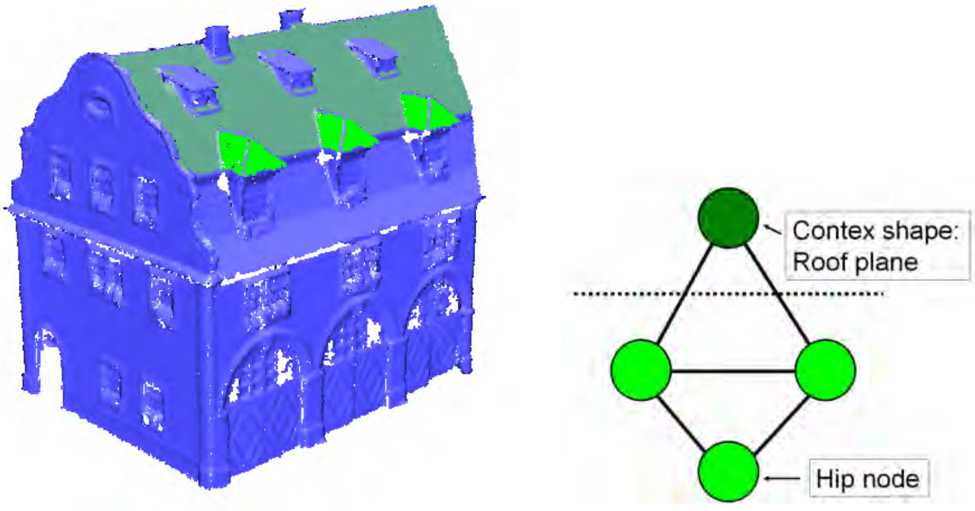
(a) Points (b) Shapes



(c) Columns

Figure 4.4: A scan of a choir screen consisting of 2M points. The query graph for the columns consisted of a cylinder connected to tori at both ends. In a) the original point-cloud is depicted. b) shows the support of the detected shape primitives in random colors. In c) the detected columns are highlighted in green.

a total of four query graphs, that only marginally differ, would have to be defined separately by the user. Since in practice this additional work may become quite burdensome, we augment the query graphs with what we call *optional nodes*. These nodes may be ignored by the matching procedure if it is unable to find any suitable counterparts in the given topology graph. To incorporate optional nodes, the matching procedure of Sec. 4.5.2 is extended in the following manner: First the matching is performed in the same way as described above, but ignoring any optional nodes. Then, for each matched instance of the query graph, as many



(a) Matching (b) Query graph

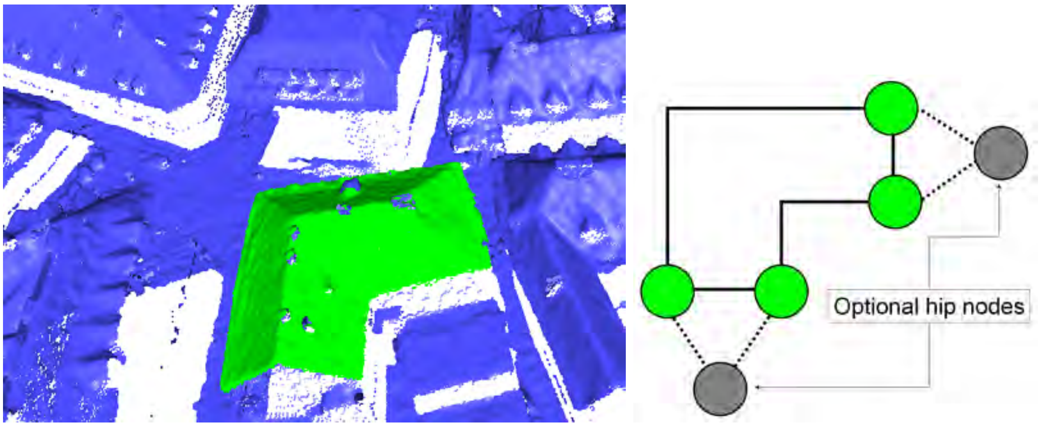
Figure 4.5: a) Detection of dormers on a roof. The roof plane shown in darker green is a context shape of the dormers. b) The query graph containing a context node.

optional nodes as possible are matched. To this end the graph traversal examines all possible matchings of the optional nodes but returns only those with the largest number of matches.

A problem arises if there are optional query graph components, i.e. sets of query nodes that should either be matched entirely or not at all, instead of only single optional nodes. In such a case simply declaring all the nodes in question as optional could lead to incomplete matchings of the component. Therefore, we use a graph constraint which asserts the completeness of the matching.

Although optional nodes increase the complexity of the search, they greatly reduce the number of required query graphs if different variants of a basic concept have to be detected. In Figure 4.6 the single hip of an L-shaped roof was matched by an optional node.

Multinodes An even more complex case arises if we want to be able to model repetitive patterns like the steps of a stairway. In order to be able to model stairways with an arbitrary number of steps, a generic way of model extension is necessary. A simple approach is to define *multinodes* in the query graph that may match *several* different topology graph nodes. A multinode is defined as a query graph node that has a self-loop, i.e. an edge connecting the node with itself (this edge is implicitly considered optional by the matching algorithm). Via multinodes



(a) L-shaped roof (b) Query graph

Figure 4.6: An L-shaped roof may be hipped on either end. This is best modeled by optional nodes in the query graph. a) A matched L-shaped roof in a stereo reconstruction of a city containing 4M points. b) The query graph used for detection. Optional nodes are shown in grey.

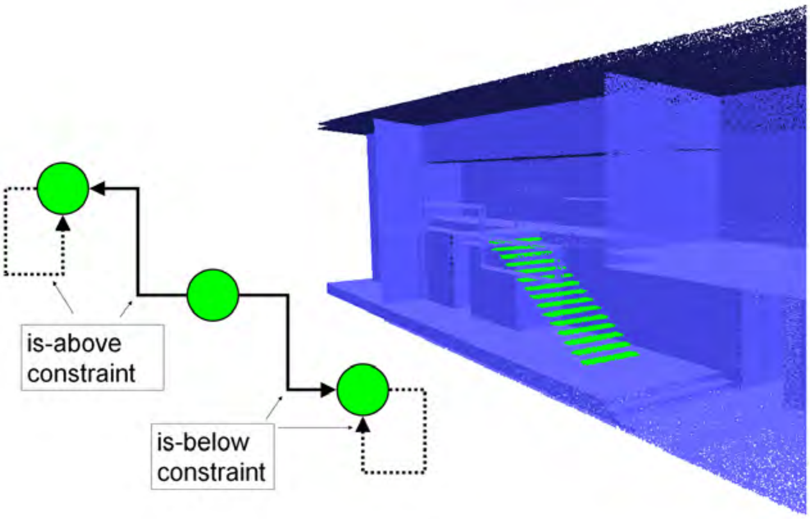


Figure 4.7: Detection of a stairway in a sampled CAD model of a house. The model was converted to a point-cloud by random sampling of the surface.

we are able to match arbitrarily large chains in the topology graph. This allows us to define a query graph for stairways using only three nodes, as depicted in Figure

4.7. Note that we make use of directed edges in this example so that the multinodes do not need to match additional neighbor nodes each time the self-loop has been traversed (since the multinode does not have an outgoing edge other than the optional self-loop).

Query refinement After the search for a query graph, the system presents the user with the results in an interactive framework that allows query refinement by changing constraints as well as query graph topology at runtime. As soon as an element of the query graph is modified, the new results are computed and displayed. This was achieved in real-time for all our tested examples.

4.6 Conclusion

Our shape detection system works on point-clouds so that we are able to work on data stemming from virtually arbitrary sources, such as terrestrial or airborne LIDAR data or stereo reconstructed scenes. Even polygon soups as well as ordinary meshes can easily be converted into a point-cloud by random sampling of the surface. As we mainly target applications in the architectural or cultural heritage domain, we safely assumed that most objects under consideration can be well represented by a set of primitive shapes. Thus we can employ the fast primitive shape extraction method from Chapter 2 to effectively reduce the redundancy in the point-cloud and to derive a concise shape representation consisting of a topology graph on the shape primitives. In this graph, our system allows the detection of features that can be described as compositions of simple primitive shapes. Due to the simple structure of our representation it is not necessary for the primitive shape detection to output an optimal segmentation with a minimal number of primitives, nor to find the correct edges and transitions between different shapes. Only the detection of the relevant structures and their rough outlines has to be ensured.

Obviously, an inherent limitation of the method is that it is unable to deal with cases for which features cannot be defined as configurations of primitive shapes, e.g. if trying to detect ornate frescos. However, we have demonstrated that for a wide range of frequently encountered structures our approach is very well suited and is able to deliver results as expected by the user. The user is able to specify the sought structures in a general way, even permitting fuzzy search within the limits of the graph constraints and the inclusion of optional components.

A potential drawback of our method could arise if large topology graphs with a lot of nodes and edges are used and at the same time the node and edge constraints of the query graph are chosen in a way that a wrong match will not be encountered early on during the matching procedure. Since the search for subgraph

dataset	#nodes	#edges	top. graph	matching
chapel (Figure 4.3)	232	406	1.8s	< 10ms
choir screen (Figure 4.4)	537	2731	1.8s	< 10ms
dormer roof (Figure 4.5)	106	138	0.27s	< 10ms
city model (Figure 4.6)	431	351	2.5s	< 10ms
CAD-house (Figure 4.7)	160	513	3.9s	< 10ms

Table 4.1: Some statistics on test models. *#nodes* gives the number of primitive shapes detected in the point cloud (there is one node per primitive in the topology graph). *#edges* states the number of edges in the topology graph. *top.graph* lists the timings for construction of the topology graph. The last column gives the timings for matching the query graph.

isomorphisms is a NP-hard problem, the retrieval performance might degenerate. However, such pathological cases are unlikely and in practice we observe very fast response times of the system, i.e. in the order of a few milliseconds (see timings in Table 4.1).

4.6.1 Future work

Future work concerning our method should address the improvement of usability. Up to now, the query graphs and the attached constraints are defined by hand. To make this process more comfortable for less experienced users we propose the development of a graphical user interface in which query graphs and constraints can easily be defined. A further step of research would be the automatic extraction of query graphs from modeled or scanned objects by means of statistical learning. Techniques like relevance feedback could be used to further enhance the retrieval performance.

Moreover, we plan to apply our system to basic point-cloud editing operations such as copy and paste of semantic units. Another interesting avenue of future research is exploiting the ability to detect self-similarities in the data for compression. Replacing instances of a query graph by generic representations might lead to very high compression ratios.

CHAPTER 5

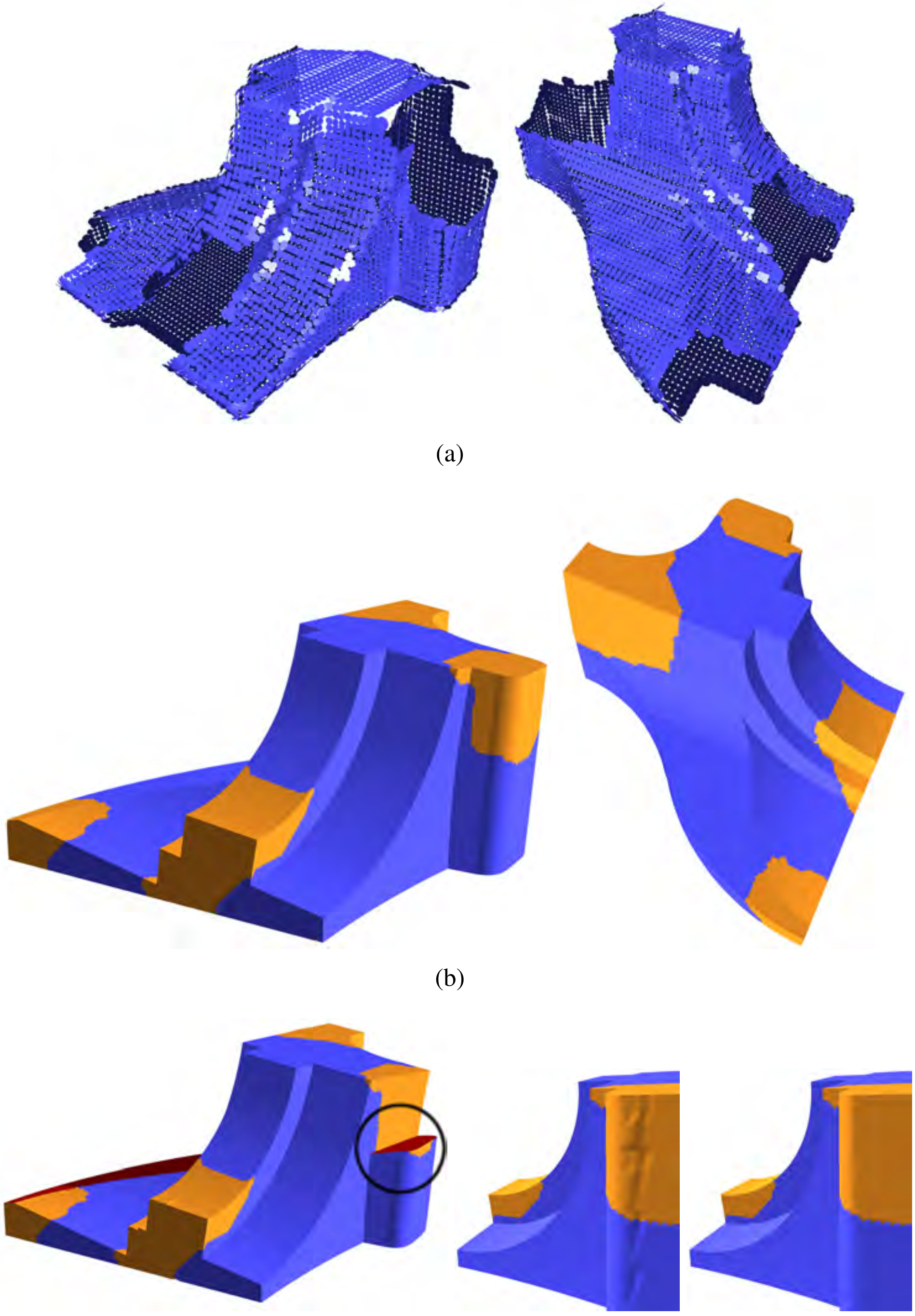
COMPLETION AND RECONSTRUCTION

5.1 Introduction

This chapter presents the final application of detected primitives examined in this work and addresses the reconstruction of incomplete models with the help of primitive shapes. While reconstruction from primitives has been the standard procedure in reverse engineering for many years [BMV01, PLH^{+}05] here we will put a special emphasis on model completion to an extent that has not been considered before. Completion of point-clouds is often necessary because, even despite considerable effort, data obtained with range scanners or stereo capture usually suffers from occluded or defective portions of objects that either could not be perceived during acquisition or have adverse material properties that hinder the scanning device. Nonetheless, a complete surface representation without holes is usually desired, and sometimes even required, for further processing or rendering. Therefore reconstruction algorithms must not only be able to recover the surface parts that have been captured, but must also synthesize plausible geometry in hole areas.

Previous work either uses general smoothness assumptions to derive the completing surface parts or relies on a database of suitable example cases from which a completing surface can be retrieved. In contrast, the method presented in this chapter is based on the fact that for a large class of objects encountered in man-made environments, surface characteristics are well represented by a set of primitive shapes (planes, spheres, cylinders, cones and tori). Besides global shape, these primitives also capture local surface differential properties. Moreover, their intersections naturally describe the structure of edges (see Fig. 5.1 and 5.2).

Thus, we propose to exploit the information given by a set of shape primitives that has been detected on the input surface to automatically infer a closed reconstruction of an incomplete 3D model. The primitive shapes can be extended into the empty regions and serve as a guidance for hole-filling. Our algorithm differs from previous work in several important aspects:



(a)

(b)

(c)

(d)

Figure 5.1: Reconstruction of the fandisk model. Orange color signifies completed surface parts. (a) The input point-cloud with holes. (b) The final result. Result without the connectivity enforcement algorithm of Sec. 5.5. The disconnected primitive highlighted in red cuts off part of the model. (d) Close-up views of result without consistent edge labels and final result (see Sec. 5.7)

- By using the primitive shapes as guidance for hole-filling, we find plausible completions that continue the geometric structure around the missing area. We can not only extend existing edges into the hole, but also derive the location of novel edges and corners in the synthesized surface from the primitives' intersections.

- Apart from completion of missing areas, our method also allows to use the guidance of primitive shapes to reconstruct a mesh that adheres to the primitives everywhere, i.e. not only in the holes. This results in an idealized and noise free reconstruction that is composed only of the primitives and ignores fine surface detail (e.g. engravings). Such a reconstruction is suitable for processing in a CAD environment.

- Additionally, our algorithm can also faithfully recover surface parts not approximated by primitives. Even fine details in the areas of the surface that are approximated by primitives can be recovered, while at the same time primitives are still used to guide the completion of holes.

Our novel algorithm is based on an energy minimization approach that allows us to infer missing geometry from a set of surrounding primitives and reconstruct a closed, idealized 3D-model. In detail, this chapter makes the following technical contributions:

- We derive a surface energy functional that incorporates the guidance given by the shape primitives. We propose a discretization that allows the application of an efficient graph-cut optimization algorithm.

- We give a novel greedy optimization strategy to minimize the above functional under the constraint that surface parts corresponding to a given primitive must be connected. This enables us to handle multiple holes in complex 3D models (see Fig. 5.1 (c)).

- We show that our method allows extraction of an idealized surface with sharp features. To this end we give an algorithm that resolves ambiguities arising from smooth transitions between primitives (see Fig. 5.1 (d)).

The proposed method is able to handle an arbitrary number of primitives that may have arbitrarily complex intersections which, to the best of our knowledge, was impossible previously. Finally, our method can also easily be adapted to the special conditions for completion of range-images or height-fields.

5.2 Previous work

The problem of hole-filling has been regarded from several different perspectives in previous work. Closely related to our setting is work on surface reconstruction and completion as well as the completion of range images. In the following the most relevant previous approaches from each area will be shortly reviewed.

Surface reconstruction involving fitted primitives has long been the standard in reverse engineering (see [BMV01]) but has also been considered in the graphics community [HDD$^+$94] [JWB$^+$06] [JKS08] [GSH$^+$07]. While these methods usually support the reconstruction of sharp features at the intersection of primitives, they do not provide any means to infer larger regions of missing geometry from the information contained in the shapes.

Our method uses energy functionals similar to the general purpose reconstruction methods of Kazhdan et al. [KBH06], Hornung and Kobbelt [HK06] as well as Lempitsky and Boykov [LB07]. However, compared to their approaches, our method is the first to incorporate information from fitted primitives and to explicitly address the problem of surface completion.

Surface completion is traditionally part of surface reconstruction algorithms (see e.g. [CL96]), but has also received research attention in its own right. Most methods have been inspired by image inpainting approaches [BSCB00] [DCOY03] and can be attributed to one of two directions of research: (1) Methods based on level-set PDEs [DMGL02] [VCBS03] and energy minimization [CDD$^+$04] (2) Example-based approaches [SACO04] [PMW$^+$08][WO02]. Methods in the first class focus on inferring smooth geometry in missing areas that also has smooth transitions to the existing surface parts. Therefore they cannot model the sharp features resulting from intersections of shape primitives. Moreover these methods cannot guarantee that for instance a hole in a cylindrical surface is also completed with the respective cylindrical geometry. The second class of methods is based on discovering (self-)similarity and regularity in or between models in order to infer the missing information from other fully captured surface parts. While example-based completion can achieve highly plausible reconstructions, it very much depends on the existence of suitable surface parts fitting into the missing area.

Podolak et al. [PR05] used a graph-cut based approach to resolve topological ambiguities of completing surfaces in 3D. In contrast to our approach no structural constraints other than smoothness can be imposed.

Mesh repair algorithms [Ju04] [Lie03] [BPK05] are able to fill small gaps in the input models but lack the capabilities to handle larger missing pieces and cannot propagate structure therein.

Range image completion has been studied by Fisher et al. in [SDF01] and [CLF02]. Similarly in spirit to our approach, albeit limited to 2D, missing regions are filled by continuation of primitive shapes that reach a hole's boundary. However, only cases of two severed boundary edges are handled. Jia et al. propose a tensor voting approach to range image completion in [JT04]. Even though some discontinuities are preserved, intersections of surfaces are not explicitly handled.

5.3 Shape primitive guided completion

Given a point-cloud with oriented normals representing the potentially incomplete surface $\mathcal{S}^0$, we seek to reconstruct a closed surface $\mathcal{S}$ that propagates the geometric structure of the original surface into the missing areas. Our idea is to represent this structure by a set of shape primitives $\Psi = \{\psi_1, \ldots, \psi_n\}$ that has been detected on the incomplete surface $\mathcal{S}^0$, see Fig. 5.2. Shape primitives are given as implicit surfaces (planes, spheres, cylinders, cones and tori) of possibly infinite extent. Using this definition of structure, the completion problem can be put as follows: Find a suitable watertight surface $\mathcal{S}$ that approximates $\mathcal{S}^0$ and at any location adheres to at least one of the primitives $\psi_i \in \Psi$. We may later relax this condition if there exist areas of the input surface that cannot be represented by primitive shapes.

While it is relatively easy to locally extend individual primitives into a hole region, intersections of multiple primitives can be highly complex and reconstruction becomes non-trivial. See Fig. 5.7 for an illustrating case where several planar primitives need to be intersected to form a complex rooftop geometry that has several sharp edge features. Our solution handles such complex intersections of an arbitrary number of primitives and gives plausible results. Other issues are caused by inexactly fitted primitives or by surface parts that could not be approximated by any primitive (cf. Fig. 5.2 (d-e) and (f-g)).

Since primitives are implicit surfaces of possibly infinite extent, it is possible that a primitive ψ_i is used to complete the surface in multiple disconnected regions that are far away from each other or dissimilar to the parts of $\mathcal{S}^0$ originally approximated by ψ_i. As demonstrated in the bottom row of Fig. 5.2 this can lead to undesirable completions or shortcuts. We therefore further require that all parts of the surface $\mathcal{S}$ following a primitive ψ_i should be connected.

In summary there are two characteristics that make up our primitive guided reconstruction: *Primitive adherence* makes the reconstructed surface follow the input primitives (see Sec. 5.4). Adherence is always in effect in hole areas, but surface parts that cannot be represented by primitive shapes at all as well as fine details can be handled using a traditional reconstruction method as outlined in Sec. 5.6. *Primitive connectivity* ensures that surface parts corresponding to a single primitive form a connected subset of the reconstructed surface (see Sec. 5.5) which, as argued above, is necessary for plausible reconstructions in case of mul-

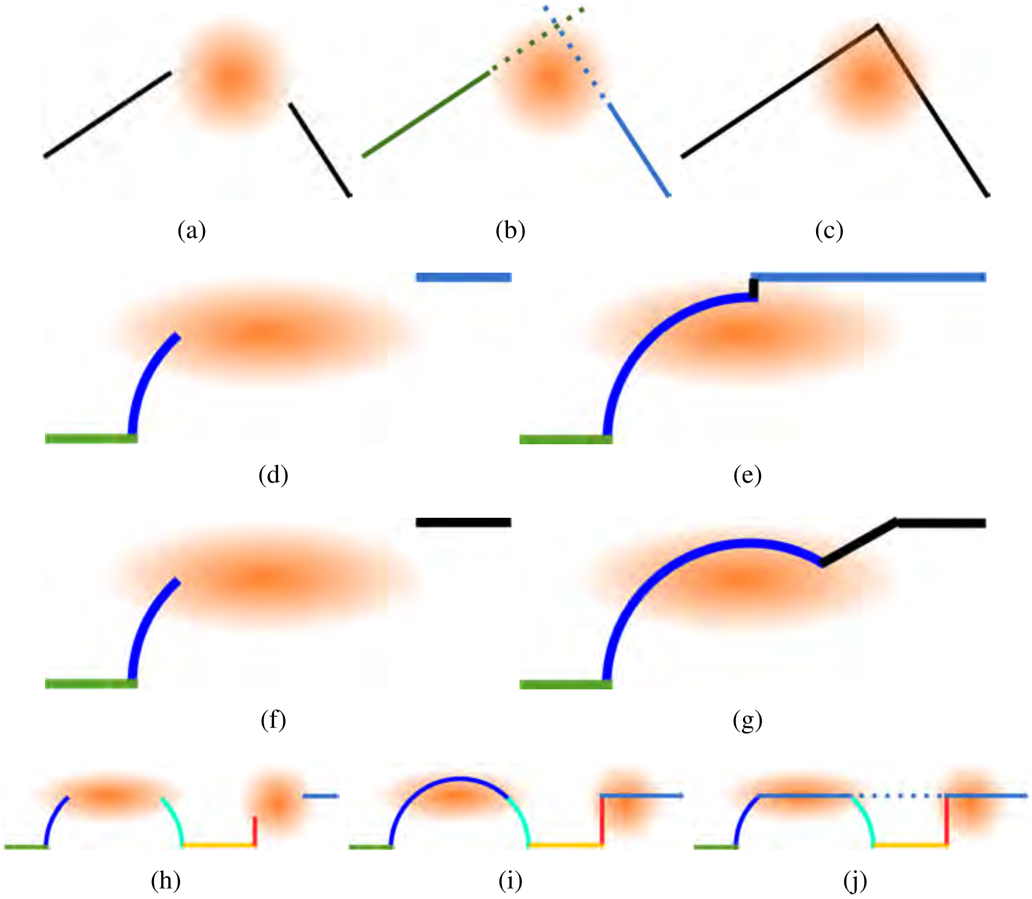


Figure 5.2: (a) A hole is indicated by the orange area. (b) Planar primitives in the vicinity are extended. (c) Their intersection defines the completed surface. (d-e) Surface areas approximated by primitives are colored. Black surface parts do not correspond to any primitive. Due to inexact primitive estimation, holes cannot always be closed with primitives only and gaps may need to be filled by other means. (f-g) If primitives are not detected for some reason (e.g. due to noise noise or because the surface cannot be approximated by primitives at all) the algorithm should nonetheless use the information of the available primitives and plausibly connect the synthesized surface to the unapproximated areas. (h-j) The effect of the connectivity constraint. (h) Primitives are color coded. (i) The desired reconstruction contains a completed circle. (j) If connectivity of the primitives is not enforced another reconstruction that cuts off the circle is also possible.

tiple holes.

5.3.1 Shape primitive detection

We use the shape primitives detected by the method given in Chapter 2. Even though the detection requires normal information, the correctness of normals close to sharp edges is not critical for the estimation of shape primitives. In our reconstruction, sharp features are deduced from the robustly fitted primitives directly, without relying on potentially noisy and incorrect normal information.

Recall that the shape detection provides us with a set of primitive shapes $\psi_i \in \Psi$ where each primitive ψ_i is associated to a single connected support area $S_i \subset \psi_i$ that corresponds to the region of $\mathcal{S}^0$ approximated by ψ_i. In general the primitives computed by the method described in Chapter 2 are unoriented such that for instance the normal of a detected plane can point either outwards or inwards. Our algorithm, however, will require oriented primitives. Therefore we derive the orientation of primitives from the per-point normals of their support region. This of course implies that we are given oriented per-point normals - which is generally not required by the primitive detection method. Fortunately, in practice the orientation of per-point normals can usually be deduced from the location of the scanning device, i.e. normals should always point towards the scanner.

While in practice shape primitives ψ_i approximate the original surface only up to a predefined tolerance, for the sake of simplicity the following discussion assumes that the surface $\mathcal{S}^0$ is given as the union of the support sets, i.e. $\mathcal{S}^0 = S_1 \cup S_2 \cup \ldots \cup S_n$. Thus we also have $S_i \subset \mathcal{S}^0$. We postpone the discussion of how details within the tolerance threshold and parts of the surface not covered by primitives can also be considered to Sec. 5.6.

5.4 Primitive adherence

In this section, we propose an energy functional that assigns any given closed surface $\mathcal{S}$ a cost according to how well the surface adheres to the given set of shape primitives Ψ. By minimizing this cost over the set of all closed surfaces we obtain a reconstruction $\mathcal{S}$ that is guided by Ψ and satisfies the primitive adherence condition. For now we will disregard the connectivity condition and postpone the discussion of enforcing connectivity to Sec. 5.5.

We define the cost E of $\mathcal{S}$ by measuring the surface area E_a of $\mathcal{S}$. To reward primitive adherence we do not take into account the area E_p of those surface parts that coincide with a primitive. A third term E_c avoids ambiguities in the solution and enforces the approximation of the original surface $\mathcal{S}^0$. Thus, denoting the surface normal of $\mathcal{S}$ by n, the functional responsible for primitive adherence that

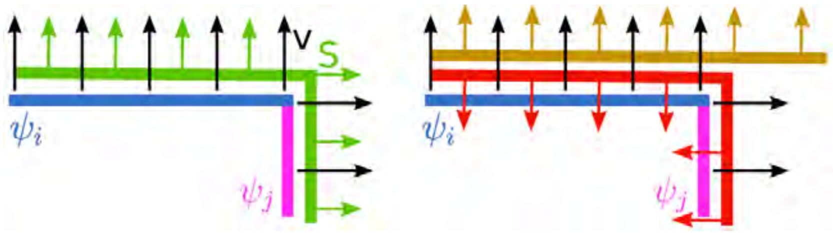


Figure 5.3: The effect of different configurations on the energy term $-\int_S \mathbf{H}(\langle n|v\rangle)dA$ (see text).

we want to minimize can be written as

$$\begin{aligned} E(\mathcal{S}) &= E_a(\mathcal{S}) - E_p(\mathcal{S}) + E_c(\mathcal{S}) \\ &= \int_{\mathcal{S}} dA - \int_{\mathcal{S}} \mathbf{H}(\langle n|v\rangle)dA + E_c(\mathcal{S}) \end{aligned} \tag{5.1}$$

where $v : \mathbb{R}^3 \to \mathbb{R}^3$ is a vector field derived from the shape primitives' normals and $\mathbf{H}$ denotes the Heaviside function

$$\mathbf{H}(x) := \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

Since the first term E_a in Eq. (5.1) is simply the surface area, minimizing E in the absence of, or far away from, primitive shapes results in a surface of minimal area.

Concerning the vector field v in the second term E_p, we compute for each primitive ψ_i the normal field $n_{\psi_i} : \mathbb{R}^3 \to \mathbb{R}^3$ which vanishes everywhere except on the surface of ψ_i. We define v as the sum over the normal fields n_{ψ_i} of all primitives. The idea behind this definition is illustrated in the left of Fig. 5.3. As shown in the figure, the vector field v (shown in black) is non-zero only on the shape primitives. Moreover, the term $\mathbf{H}(\langle n|v\rangle)$ evaluates to 1 iff the orientation of the surface normal field n (shown in green) and v coincide. For the configuration shown on the left of the figure both criteria are met. In this case, the second term cancels the area term resulting in zero cost. The sum of the terms increases if the surface does not follow the primitive shapes (yellow on the right) or if surface normal and vector field are not consistently oriented (red surface). Thus, minimizing the second term in the above energy aligns the surface with primitive shapes and enforces the correct orientation of $\mathcal{S}$.

Even though the first two terms of the cost function enforce adherence to the primitive shapes ψ_i, there are usually multiple minima, including the trivial empty surface. Therefore, the third term E_c of the function E is used to impose additional inside and outside constraints given by two sets $C_{in} \subset \mathbb{R}^3$ and $C_{out} \subset \mathbb{R}^3$. As a closed surface $\mathcal{S}$ divides the space into a well defined inside $\mathcal{S}_{in}$ and outside $\mathcal{S}_{out}$ (see Fig. 5.4(b)) and we require $C_{\{in,out\}}$ to be contained in the inside/outside $\mathcal{S}_{\{in,out\}}$ respectively. The actual choice of the sets $C_{\{in,out\}}$ is derived from the original surface $\mathcal{S}^0$ and detailed in Sec. 5.4.2. The third term in the cost function which integrates over all violated constraints is given by

$$E_c(\mathcal{S}) = \int_{C_{in} \setminus \mathcal{S}_{in}} \lambda dV + \int_{C_{out} \setminus \mathcal{S}_{out}} \lambda dV \tag{5.3}$$

where λ is a constant that must be chosen sufficiently large to avoid constraint violations, i.e. larger than the area of the reconstructed surface.

5.4.1 Discrete global minimization

While the cost function E enables an adequate formalization of the surface completion problem as sketched in Sec. 5.3, the multitude and structure of its local minima hinders an efficient global minimization by variational methods. To enable an efficient global optimization, we pursue an approach similar to Kolmogorov and Boykov [KB05] and formulate the surface completion problem defined in the previous section as a cut on a discrete volumetric graph. Fast graph-cut methods can then be used to compute a globally optimal solution in polynomial time.

We start by defining a volumetric graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the set of vertices $\mathcal{V}$ consists of all voxels in a regular $3D$-grid that bounds the input surface S^0. In addition, $\mathcal{V}$ contains a special source vertex s and a sink vertex t. The edges in $\mathcal{E}$ connect each grid node v_{ijk} to its 26 neighbors in the grid. The definition of the volumetric graph is illustrated in Fig. 5.4 (a) for a $2D$ example.

On the edges of this graph we define a capacity function $\hat{E}$, that assigns each directed edge a cost. This function $\hat{E}$ can be regarded as a discrete counterpart to the continuous cost function E in the previous section. Given the graph $\mathcal{G}$ and the discrete cost function $\hat{E}$, an optimal partition of the vertices into two sets S_{in} and S_{out} - the cut - is computed by minimizing the costs of all edges from S_{in} to S_{out} subject to the constraints $s \in S_{in}$ and $t \in S_{out}$. With a suitable choice of the cost function $\hat{E}$, the discrete solution to the surface completion problem is then implicitly defined by the cut (S_{in}, S_{out}) (see Fig. 5.4 (b)).

Corresponding to the three terms of the continuous cost function E, we define the discrete edge costs assignment $\hat{E}$ as the sum of three functions

$$\hat{E} = \hat{E}_a - \hat{E}_p + \hat{E}_c \tag{5.4}$$

that constitute discrete measures of surface area, primitive adherence and constraint violations respectively. For the area costs $\hat{E}_a$ we resort to the weighting

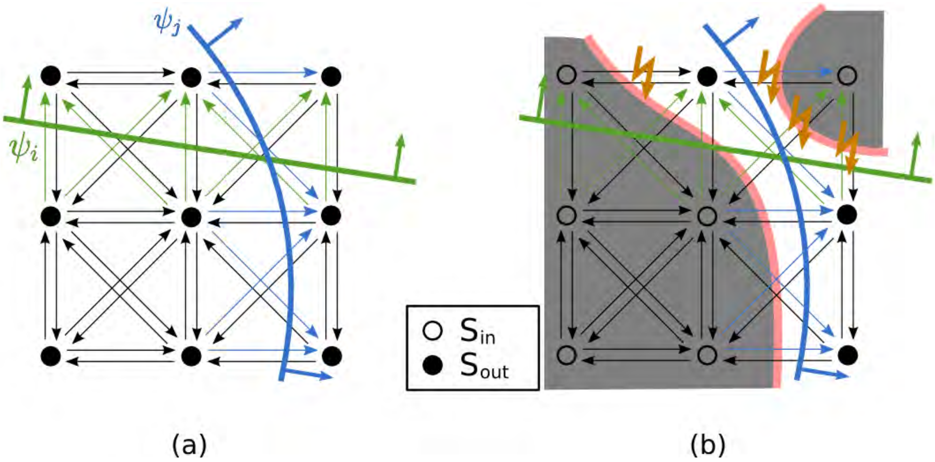


Figure 5.4: (a) Graph construction and cost assignment: The colored edges intersect a primitive and match its orientation. On these edges $\hat{E}_p$ is set to cancel their area costs. (b) High costs result, if a cut does not follow primitives or fails to match their orientation (flashes mark edges with high costs).

scheme given in [BK03b] which provably converges to the continuous area term for a growing number of grid neighbors connected to each voxel.

To define a discrete analogon of the second term in Eq. (5.1), for each directed edge $e \in \mathcal{E}$ the set of intersecting primitives is computed. If the orientation of any intersecting primitive ψ_i is consistent with the direction of e, we set $\hat{E}_p(e)$ to $\hat{E}_a(e)$ so that it cancels the area term's contribution (see Fig. 5.4(b)). Formally, we set

$$\hat{E}_p(e) := \begin{cases} \hat{E}_a(e) & \text{a } \psi_i \text{ intersects } e \text{ and } \langle n_{\psi_i} | e \rangle > 0 \\ 0 & \text{otherwise} \end{cases}$$

where n_{ψ_i} denotes the normal of ψ_i at the intersection point. This choice of $\hat{E}_p$ does indeed mimic the behavior of the second term in the continuous formulation (5.1) in the following sense: For a cut (S_{in}, S_{out}) as shown in Figure 5.4 high cost results at edges that are not intersected by any primitive or if the orientation of primitive and cut does not match. Therefore, a cut minimizing $\hat{E}_p(e)$ adheres to shape primitives and the orientation of the resulting surface S matches that of the followed primitives.

Just as in the continuous case we need to add a third term $\hat{E}_c$ to enforce certain inside and outside constraints derived from the input surface S^0. In the discrete setting, we assume that C_{in} and C_{out} are given as sets of vertices corresponding to voxels in the inside and outside respectively. In analogy to the continuous case, $\hat{E}_c$ should be large if either $C_{in} \subset S_{in}$ or $C_{out} \subset S_{out}$ is violated. As by definition of the graph cut we have $s \in S_{in}$ and $t \in S_{out}$ it is sufficient to add edges with high costs connecting s to vertices in C_{in} and likewise connecting vertices in C_{out} to t. More precisely, the cost function $\hat{E}_c$ is defined to zero on all but these extra edges to which it assigns the high constant cost λ (in practice the maximal representable value of the employed data type is a viable choice). A violation of e.g. an inside constraint $v \in C_{in}$ will therefore result in a cut through this extra edge and thus in high overall costs. In the following section we will discuss the actual choice of the sets C_{in} and C_{out} .

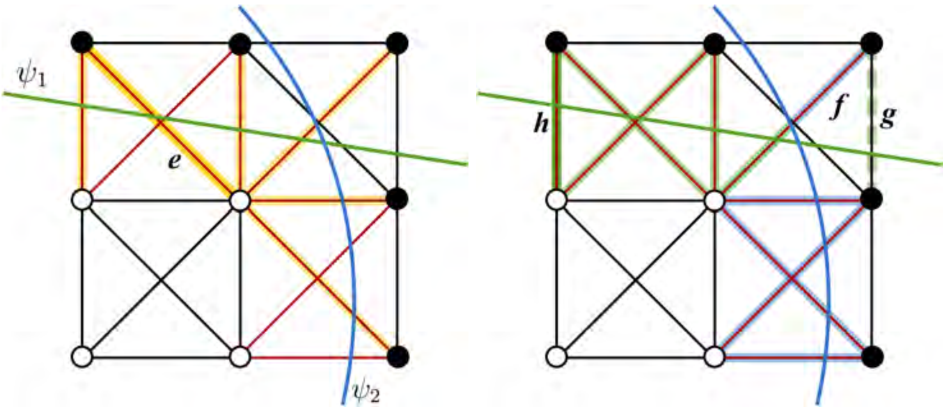
The above discrete formulation of surface completion is closely related to the cut metric of Kolmogorov and Boykov [KB05] who proved for the $2D$ case that all functionals representable by cut metrics are of the form in Eq. (5.1). Although the construction used in the proof generalizes to $3D$, there are some ambiguities in the choice of the edge cost assignment. Therefore the choice of an optimal edge assignment is not clear, in particular if sharp features at primitive intersections are to be preserved. The discrete formulation presented here is directly adapted to primitive shapes and thus circumvents this problem. Moreover, it establishes an explicit edge to shape correspondence which is crucial for enforcing the connectivity constraint described in Sec. 5.5.

In general, a drawback of the volumetric approach is the huge memory demand of the 3D-grid graph. However, recently Lempitsky and Boykov [LB07] proposed a hierarchical graph-cut approach that guarantees global optimality while operating only on a banded subset of the volume. Their technique is also applicable in our setting and we use it for completion of large models.

5.4.2 Placement of inside and outside constraints

So far, the original support of primitives has not been considered in the definition of $\hat{E}$. We therefore propose a simple scheme to derive constraints C_{in} and C_{out} from the support sets S_i of all primitives. In case of a single oriented primitive ψ_i, we start by computing the set of all directed edges E_{S_i}, that intersect the support S_i of ψ_i. We only consider those edges that run from the inside of ψ_i to the outside, i.e. that match the orientation of ψ_i. Then a natural choice for the inside constraints C_{in} consists of the set of voxels from which an edge in E_{S_i} emanates. Appropriate outside constraints can be defined in a similar fashion.

In the general case, defining inside and outside constraints in this way can lead to contradictions at primitive intersections. We therefore add a vertex v with an emanating edge in E_{S_i} to C_{in} only if it is not contained in E_{S_j} for any other primitive ψ_j.



(a) (b)

Figure 5.5: Edge connectivity and edge/primitive correspondence. Cut-edges are depicted in red. (a) The cut-edges connected to edge e are highlighted. (b) Cut-edge f is intersected by both primitives. If h is part of the original support S_1^0 of ψ_1 and the cost of g had previously been increased, it is now reset because it is connected to h, see Sec. 5.5.

5.5 Primitive connectivity

The above algorithm does not enforce connectivity of the areas $S_i \subset \mathcal{S}$ corresponding to primitive ψ_i. However, as outlined in Sec. 5.3 and illustrated in Figures 5.2 and 2.2 this is often crucial. From the shape detection we do have a connected region $S_i^0 \in \mathcal{S}^0$ for each primitive ψ_i where it is supported by the original surface. Thus we require the following: The area S_i corresponding to ψ_i in the reconstructed surface $\mathcal{S}$ is a connected superset of S_i^0.

Unfortunately this connectivity constraint cannot be formulated as a graph-cut problem, see e.g. the work of Kolmogorov and Zabin [KZ04] who give a good characterization of functions minimizable by graph-cuts. Instead, in order to find the global optimum an involved combinatorial optimization is necessary which quickly becomes infeasible with a growing number of shape primitives. We therefore suggest a less complex but nonetheless effective iterative greedy optimization scheme.

This iterative optimization makes use of the graph structure described in the previous section. After the graph-cut, the reconstructed surface is implicitly defined by the set of cut-edges, i.e. all halfedges running from S_{in} to S_{out}. Since our graph construction provides an explicit edge/primitive correspondence, we can use the cut-edges to determine connectivity on the surface and to efficiently identify correspondence between surface parts and primitives. In fact we treat each cut-edge as a representative for a small local patch in the reconstructed surface.

For each cut-edge we identify the set of intersecting shape primitives in order to establish surface/primitive correspondence, i.e. if a cut-edge is intersected by primitive ψ_i the respective surface patch is part of S_i, see Fig. 5.5 (b). We say two edges are connected (and therefore also their respective surface patches) if they share a common node in the graph, see Fig. 5.5 (a). With these definitions the connectivity condition can be restated as follows: The cut-edges corresponding to primitive ψ_i must form a connected superset of the cut-edges intersected by S_i^0. We call any cut-edge corresponding to ψ_i that does not belong to this connected set a violating edge.

The basic idea of our connectivity enforcing algorithm is in each iteration to greedily set the cost of all violating cut-edges equal to the cost given by $\hat{E}_a$. Then the graph-cut is re-run with the increased cost on the violating edges. This means that the newly reconstructed surface will avoid the now costly edges and prefer cheaper edges corresponding to other primitives. In order to remedy some of the greedy decisions of earlier iterations, the cost of graph edges whose cost had previously increased but are now connected to non-violating cut-edges is reset in each iteration, see Fig. 5.5 (b). The procedure is repeated until the set of cut-edges does not change between iterations.

In summary, the iterative optimization consists of the following main steps: (1) Compute the reconstruction using the graph-cut algorithm (2) Detect violations of the connectivity constraint by inspection of the cut edges. Since S_i must contain S_i^0 the non-violating edges can be identified by a graph traversal visiting all cut-edges connected to S_i^0. (3) Increase cost of violating edges in the graph and reset cost of revalidated edges (4) Reiterate until the set of cut-edges converges. Although convergence cannot be guaranteed in general, we found that in practice all of our test cases converged in less than fifteen iterations.

5.6 Reconstruction of detail

In the previous sections we have presented an algorithm that uses guidance from primitives for hole-filling and gives an idealized reconstruction of the input surface that adheres to the primitives everywhere. Such a reconstruction is often useful if the model is to be used in CAD systems or if the input data was corrupted by many outliers and noise. However, high quality scans may contain valuable detail geometry, e.g. engravings, or models may contain parts that cannot by approximated by primitives at all, e.g. a small statue mounted on a wall. Depending on the application it may be desirable to recover these features as well and in this section we outline how we can seamlessly combine our primitive based reconstruction with the one given by Lempitsky and Boykov [LB07] to this end.

Lempitsky and Boykov define a smooth vector field u from a set of input points

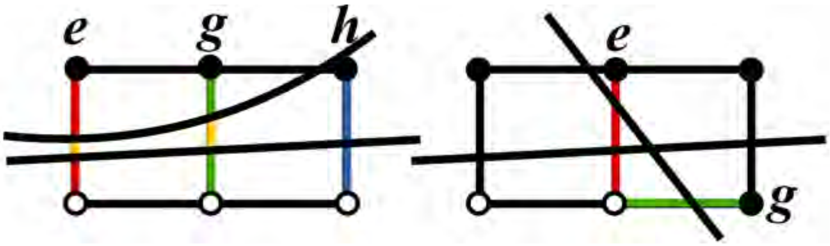


Figure 5.6: Illustration of the different cases for the smoothness term V in eq. (5.8). Left: $V_{e,g}$ is of type (2) and will give the distance between the primitives along edge e, while $V_{g,h}$ is of type (3) and gives the distance between the intersections of g. Right: $V_{e,g}$ is of type (3) and evaluates to zero because the primitives intersect within the cube face.

with oriented normals (please see the original paper for details). In our setting we can either use the entire point-cloud for computation of u or, if we are certain that the input model is in principle well represented by primitives, we can use only the points associated to the primitives and ignore any remaining points as noise and outliers. Thus, given the vector field u the detail preserving reconstruction functional is stated as follows:

$$E(\mathcal{S}) = E_a(\mathcal{S}) - E_p(\mathcal{S}) - \lambda E_u(\mathcal{S}) \tag{5.5}$$

where E_a and E_p are as in Eq. (5.1) and

$$E_u(\mathcal{S}) = \int_{\mathcal{S}} \langle n|u\rangle \, dA \tag{5.6}$$

In contrast to E_p we can apply the divergence theorem to Eq. (5.6) because u is smooth, such that

$$E_u(\mathcal{S}) = \int_{\mathcal{S}_{in}} div(u) dV \tag{5.7}$$

Note that we no longer include the $E_c(\mathcal{S})$ from Eq. (5.1) since adherence to the original surface is now ensured by the term $E_u(\mathcal{S})$. Except for the missing in/out constraints, the graph construction of Sec. 5.4.1 remains unchanged. However additionally, $div(u)$ is evaluated on the grid nodes and depending on the sign s- or t-links are added with a cost proportional to the divergence, please refer to [KB05] for details.

5.7 Surface extraction

After application of the algorithms of the previous sections, all voxels have been classified as either inside or outside and the final task is to extract the resulting surface mesh. Of course it is possible to extract a mesh using the standard marching cubes algorithm [LC87], but this would not faithfully recover the shape primitives and does not capture any sharp features at the intersections. Instead, we can exploit the tight coupling between cut-edges and primitives discussed in the previous sections to employ the extended marching cubes algorithm of Kobbelt et al. [KBSS01] for recovery of shape primitives as well as sharp features.

In order to recover sharp features, the extended marching cubes requires for each cube edge intersected by the surface not only the point of intersection but also the surface normal at that position. This information is easily obtained in our setting: If a cut-edge is labeled with a shape primitive ψ_i then the point of intersection as well as the normal are computed using ψ_i. If a cut-edge is not associated with any primitive then the midpoint of the edge is taken as intersection point and the normal is ignored.

However, it can happen that a single cut-edge is labeled with two or more primitives, see e.g. Fig. 5.5. This is usually the case near primitive intersections or locations where primitives come close to each other, see also Fig. 5.1. These situations need to be disambiguated in order to achieve high quality results. For the disambiguation only the cut-edges that will actually be inspected by the extended marching cubes algorithm need to be considered, i.e. only the axis aligned edges of the cubes and no diagonals. We will denote this set of cube edges by $\mathcal{E}_c$. We will also need a neighborhood relation $\mathcal{N} \subset \mathcal{E}_c \times \mathcal{E}_c$ between the edges in $\mathcal{E}_c$ which is different from the one defined in Sec. 5.5 as this time the diagonal edges are missing. In the following, two edges in $\mathcal{E}_c$ are said to be neighbors if they are adjacent to a common cube face.

5.7.1 Consistent edge labeling

Our algorithm for derivation of consistent edge labels is based on the following observation: Neighboring edges should have different primitive labels only if the two primitives come very close or intersect in the space between the edges. Otherwise continuation of a primitive should be preferred. Such label dependent neighbor relations lead to a well studied class of energy functions of the following type[1] (see [SZS⁺08] for a survey):

$$C(\mathbf{f}) = \sum_{e \in \mathcal{E}_c} D_e(f_e) + \sum_{e,g \in \mathcal{N}} V_{e,g}(f_e, f_g) \tag{5.8}$$

where $\mathcal{E}_c$ is a set of sites, in our case the set of edges introduced above. $\mathbf{f}$ is a labeling of the sites, i.e. a mapping from $\mathcal{E}_c$ to Ψ. Thus, $\mathbf{f}$ assigns each edge a

---

[1] Interestingly, the discretization in Sec. 5.4.1 can also be interpreted as a realization of such a functional, but the motivation from a geometric point of view is much more intuitive. See [KB05] and [KZ04] for an in-depth discussion of the relationship.

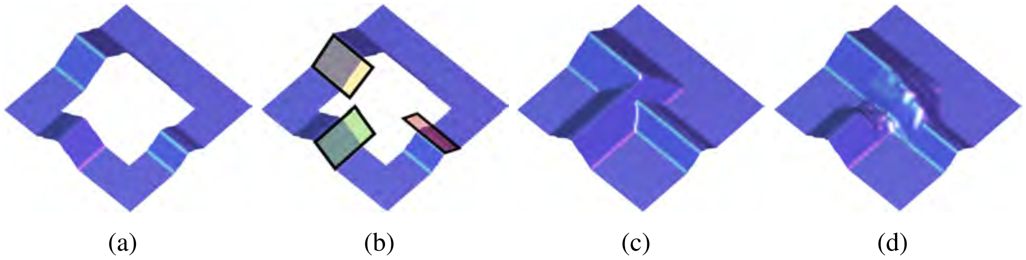


Figure 5.7: (a) Height field with missing area. (b) Planar primitives are detected on the ridges (c) Our result (d) Completion result obtained by [JT04]

primitive from Ψ. $D_e(f_e)$ is a data cost term that specifies the cost of assigning label f_e to edge e and is used to restrict the set of possible labels for the edge e. $V_{e,g}$ measures the cost of assigning the labels f_e, f_g to the adjacent edges e, g and is responsible for ensuring the above conditions on shape changes between edges. The term $D_e(i)$ vanishes if e is among the set of edges associated to primitive ψ_i, otherwise we set $D_e(i) = \infty$. $V_{e,g}(i,j)$ vanishes if $i = j$, otherwise we distinguish the following cases (see also Fig. 5.6):

(1) One of the labels is invalid for the respective edge, i.e. $e \notin S_i$ or $g \notin S_j$. In this situation the cost of $V_{e,g}$ is meaningless as the data term D_e will be infinite and we simply set $V_{e,g}$ to zero.

(2) Both labels are valid on both edges. If shapes are approximately tangent, the cost should be lowest at the point where both primitives are closest to each other. If the shapes intersect within the cube we let $V_{e,g}$ vanish, otherwise we let

$$V_{e,g}(i,j) = \min(\|I_e(i) - I_e(j)\|, \|I_g(i) - I_g(j)\|) \tag{5.9}$$

where $I_e(i)$ denotes the point of intersection of e and ψ_i. See Fig. 5.6 on the left.

(3) One edge is valid for both labels while the other can only be assigned one of the labels. Here the same arguments hold as in case (2) and $V_{e,g}$ vanishes if the primitives intersect, otherwise we let $V_{e,g}(i,j) = \|I_h(i) - I_h(j)\|$ where $h \in \{e, g\}$ is the edge intersected by both primitives. See Fig. 5.6 on the right.

We use the graph-cut based algorithm of Boykov et al. [BVZ01] for optimization of Eq. (5.8). While this method is quite efficient, performance can be increased by optimizing different connected components of ambiguous edges separately.

5.8 Height-fields

Our algorithm can directly be applied to geometry derived from height-fields, but the generated completion might not be representable as the graph of a function

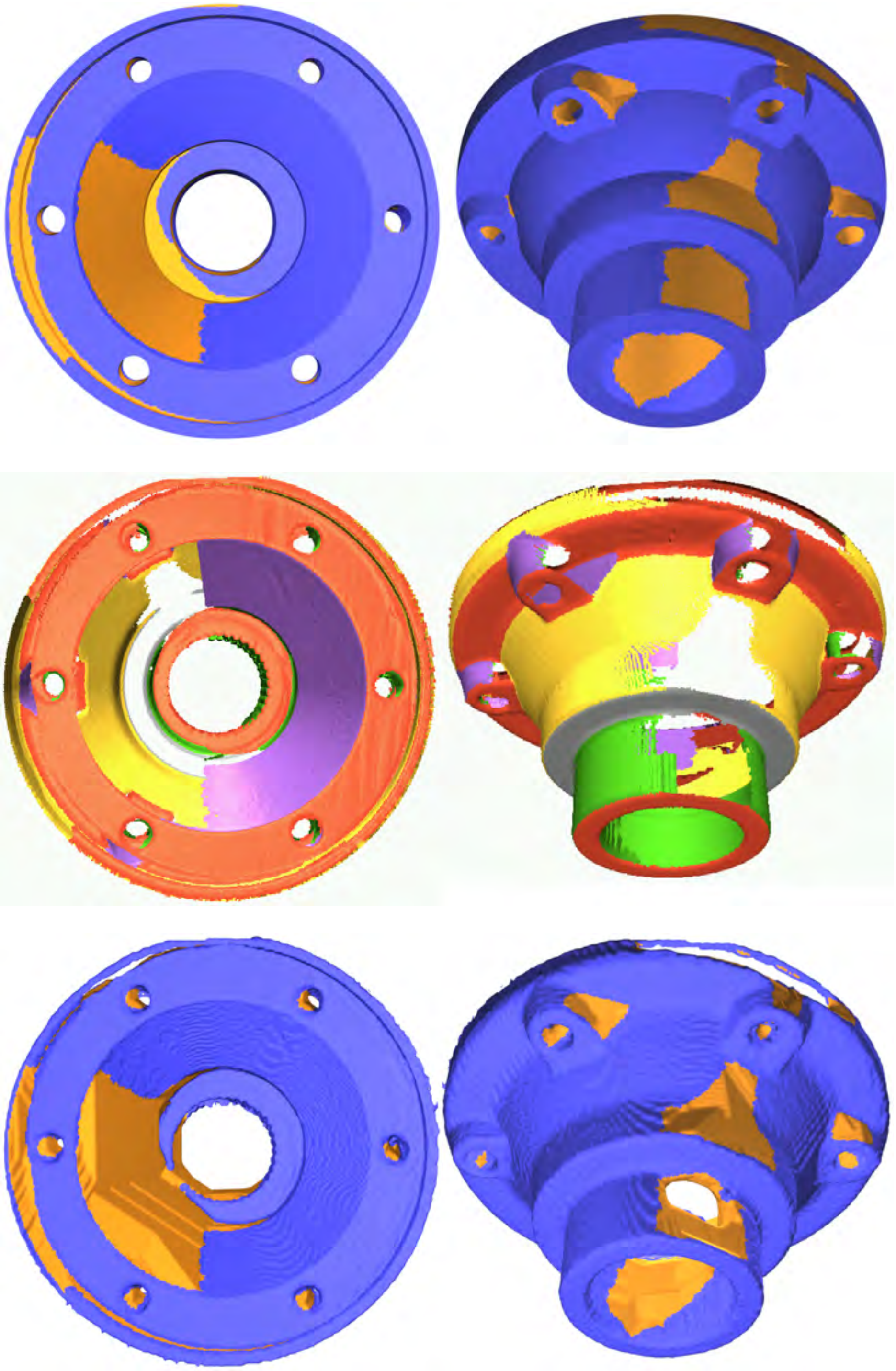


Figure 5.8: Completion of the carter model. Orange color signifies completed surface parts. Top row: Our final result with sharp features. Middle row: The input point-cloud with holes. Primitive types are colored as follows: plane/red, sphere/yellow, cylinder/green, cone/purple, grey/torus. Bottom row: The result with the algorithm of [LB07].

over the ground. To ensure this we extend the graph-cut construction given in Sec. 5.4.1. We only need to prevent cuts that produce switches from outside to inside in direction of the height-axis. To this end it suffices to add maximal weight to each directed edge $(v_{i,j,h}, v_{i,j,h-1})$ connecting a node with its neighbor below. This follows from the graph rules given in [KZ04], a similar construction was also used by Rubinstein et al. [RSA08].

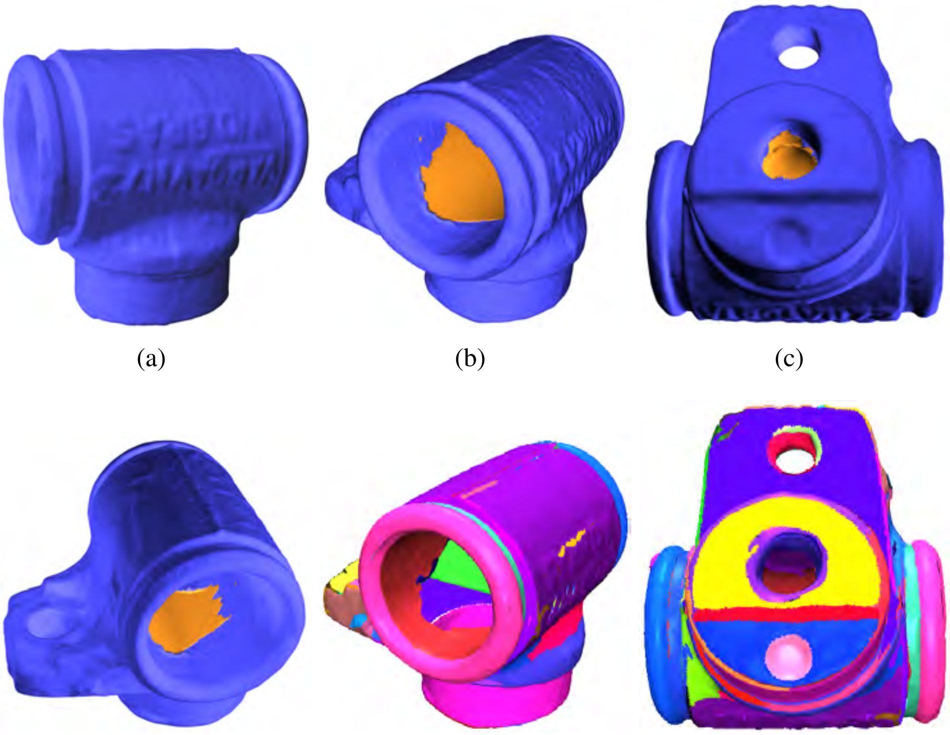
For completion of height-fields we also allow the detection of primitives on depth discontinuities by insertion of additional point samples on the surfaces implicitly spanned by these discontinuities. Shapes on depth discontinuities will allow propagation of the discontinuities into the empty region during hole-filling (see e.g. Fig. 5.10).

5.9 Experimental results

To illustrate the ability of our method to handle complex intersections of primitives we show a synthetic example of an incomplete height field in Fig. 5.7. The result of the range-image completion method proposed by Jia and Tang [JT04] is contrasted to the one of our approach. Our method produces a plausible result even in this complex case, while the other algorithm fails to reconstruct any sharp features and does not prolong the planar surface parts of the height-field.

Another synthetic example is given in Fig. 5.1. We manually removed several parts of the fandisk point-cloud and applied our algorithm. In this Figure, the effects of both the connectivity enforcing as well as the consistent edge labeling can be observed. In (c) the reconstruction result without connectivity enforcement is depicted. On the right side of the model a part of the surface has been cut off by a disconnected primitive. Our iterative connectivity enforcement algorithm successfully increases the cost of the violating surface parts and arrives at the final solution shown in (b). A part of the fandisk where a cylinder and a plane are almost tangential is shown in (d). Without the consistent edge labeling the transition between the primitives appears bumpy in the reconstruction ((d) left). This is because the edges considered in the marching cubes algorithm are associated arbitrarily with one of the two shapes. Our edge labeling method on the other hand finds a smooth transition ((d) right).

Results for a real-world case are shown in Fig. 5.8. Nine range scans of the carter model were registered into an incomplete point-cloud. 38 shape primitives were detected on this point-cloud (see second row). In the final reconstruction shown in the first row all holes were successfully filled using the shape primitives. Sharp features were faithfully recovered on the entire model. For comparison we also show in the bottom row the result obtained with our implementation of the algorithm of Lempitsky and Boykov [LB07], which fills holes with minimal



(a) (b) (c)

(d) (e) (f)

Figure 5.9: Completion of the master cylinder. (a)-(d): The final result using the detail reconstruction of Sec. 5.6 (e)-(f): Input point-cloud. Areas corresponding to different primitives rendered in random colors.

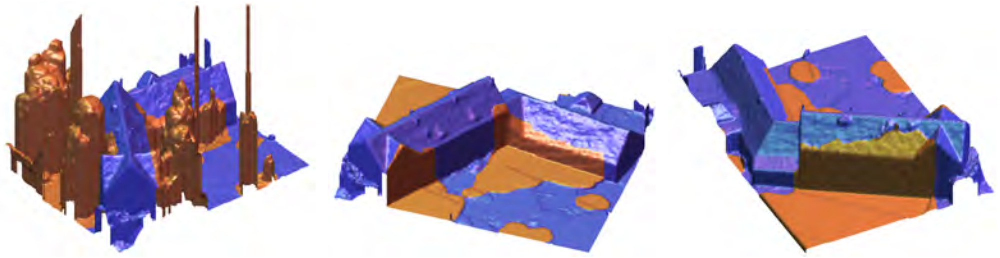


Figure 5.10: Removal of defective and undesired data from an aerial heightfield (highlighted in orange). Our algorithm infers missing features such as dormers or walls by propagating surrounding structure represented as primitive shapes.

surfaces but does not use any shape primitive guidance. While the algorithm reconstructs a watertight surface, several holes are closed in an undesired manner. Clearly, our method strongly benefits from the additional guidance provided by the primitives and is therefore able to find the correct reconstruction.

To demonstrate a reconstruction using the detail preserving variant of our method given in Sec. 5.6 we have applied our algorithm to the master cylinder model depicted in Fig. 5.9. This model contains many elements that cannot, or only very roughly, be approximated by shape primitives, e.g. the engraved writing. However the holes contained in the input model are well suited for completion with primitives. As can be seen in the images, small detail is well preserved on the original surface while at the same time the holes are plausibly filled using the primitive information.

A case of height field completion is given in Fig. 5.10. We manually removed occluding vegetation and defective elevation data from this aerial reconstruction. The missing geometry of the houses and floor were successfully reconstructed. The roof gable and the dormer were correctly inferred from the surrounding primitives.

In Fig. 5.11 we show a reconstruction result from 9 scans of the oil-pump model. The point-cloud has several large holes and is decomposed into 171 primitives. With the guidance of the primitives our algorithm finds a very plausible completion despite the significant amount of missing geometry. However, some limitations of our approach become apparent as well. In the marked area of the top right image our algorithm cannot follow the cylindrical shape of the protruding element because no primitive could be detected in the region. On the other side of the model this is not the case and the protrusion is handled correctly. Since our method has no concept of symmetry it cannot deduce the correct reconstruction from the information on the opposite side. Also tiny artifacts may occur at primitive boundaries if primitives are slightly misaligned and resulting gaps are filled with minimal surfaces (see e.g. backside of oil-pump in accompanying

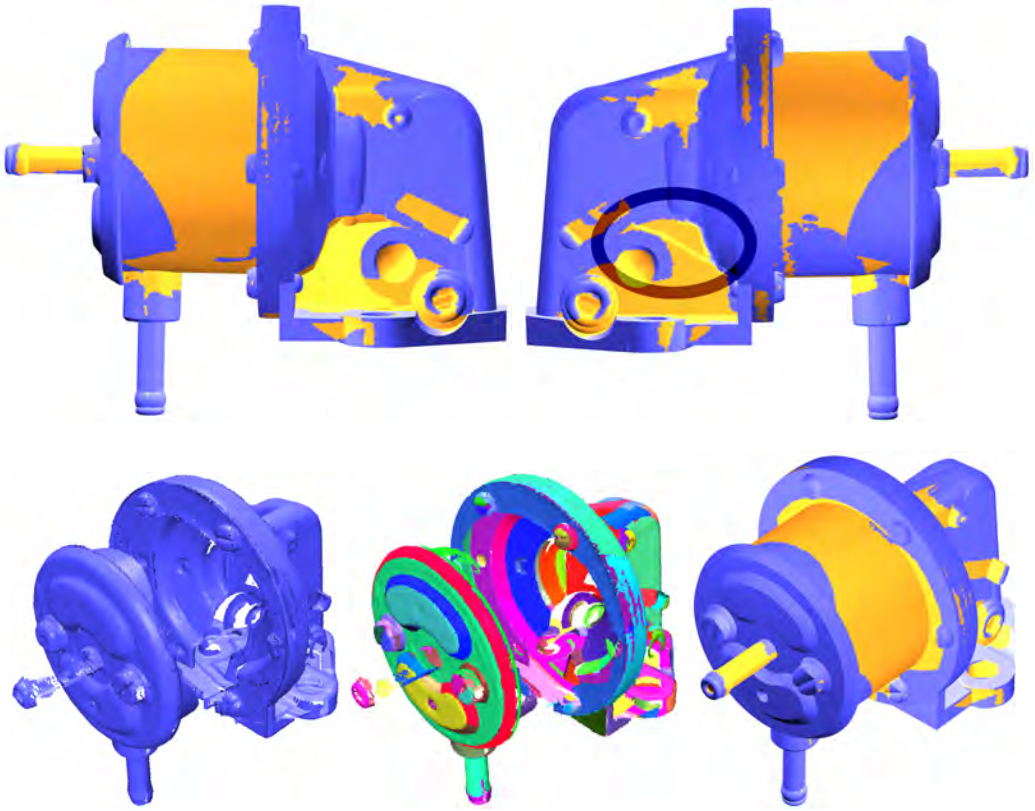


Figure 5.11: Reconstruction of the oil-pump from 9 scans. Top: Final result. Bottom from left to right: Input point-cloud. Input point-cloud colored by primitives. Final result.

model	T_s	$\|\Phi\|$	T_r	$\|V\|$
fandisk	0.27	22	1:11	193x113x199
carter	5.4	38	1:54	181x199x167
master cylinder	5.9	60	5:40	389x349x398
house	6.5	51	8:13	509x381x201
oil-pump	18.6	171	9:29	317x265x265

Table 5.1: Timings for shape detection in seconds (T_s), number of detected primitives ($|\Phi|$), timings for reconstruction in minutes (T_r), virtual size of volume ($|V|$)

video). This could be alleviated by applying a refitting algorithm as suggested in [JKS08].

Finally, we give some timings of our method in Tab. 5.1. We also give the size of the virtual grid used for the graph-cut. Although the worst-case complexity is cubic with respect to the grid resolution, the actual amount of allocated nodes is however far less than the full grid since we use only a banded subset of the volume. On our examples we observed band sizes between only 10% and 15% of the actual grid size and decreasing fractions for higher resolutions. In general the choice of grid resolution is not critical as long as voxels are small enough to separate between inside and outside areas of the volume. The precision of the reconstructed surface is hardly affected by the voxel resolution since mesh vertices are positioned exactly on the primitives. Primitive intersections are faithfully recovered due to the extended marching cubes algorithm.

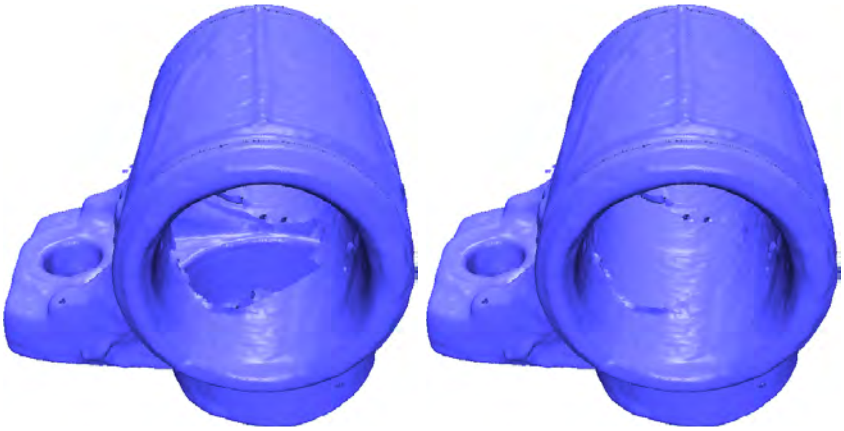


Figure 5.12: A preliminary result of detail synthesis on the completed parts of the master cylinder. The image inpainting approach of Komodakis [Kom06] is used to complete the height field over the primitive.

5.10 Conclusion

We have proposed a novel method for reconstruction of 3D-models that is guided by a set of primitive shapes and uses this guidance to complete missing parts of the input geometry. We have shown that our algorithm performs well on various involved examples with many holes on which previous methods fail to deliver correct results. While we currently employ only a small set of implicit surfaces, which is nonetheless sufficient in many cases, our method could also be extended to more complex primitives such as e.g. NURBS. However, regardless of the nature of the primitives, our algorithm is always limited to reconstructions deducible from the set of primitives that have been detected in the vicinity of the holes and in the future we plan to research a combination of our approach with methods based on self-similarity such as symmetry detection or texture synthesis, which should enable the completion of a whole new class of challenging scenarios. Moreover it is relatively straightforward to combine our method with image inpainting approaches [DCOY03, Kom06] in order to synthesize detail geometry on the completed shape primitives [BF05]. A preliminary result is shown in Fig. 5.12.

CHAPTER 6

CONCLUSION

The focus of this work is the detection and use of shape primitives for efficient point-cloud processing. While the traditional use of shape primitives has been mainly limited to the reverse engineering field, this work explored several alternative applications and developed novel algorithms for their effective use in these scenarios. The basis for the different applications is given by a novel efficient and robust primitive detection method that operates directly on the point-clouds. In summary, the following contributions have been made:

- A RANSAC based detection method has been introduced, discussed and analyzed. A theoretical analysis of the employed local sampling strategy has been given. The local sampling allows robust extraction of primitives with high probability even in large point-clouds. In conjunction with the lazy score evaluation scheme this leads to an efficient and effective algorithm.

- An algorithm for compression of the decomposed point-cloud has been presented. Due to the good approximation quality of the primitives it is possible to efficiently compress displacement maps using image-based techniques. This even allows for fast decompression on the GPU during interactive rendering.

- The segmentation and classification provided by the primitives has been exploited to automatically detect user specified entities in the point-cloud. Detection is reduced to a graph matching problem: Entities are described by their comprising primitives, represented as graph nodes, and their geometrical relations, represented by graph edges. The matching is efficient even on large point-clouds since the number of primitives is much lower than the number of points and invalid matches can be quickly pruned if geometric constraints associated with the edges are not met.

- Completion of unobserved parts of geometry has been approached by extending detected primitives in the empty areas. In order to allow the completion of even very complex holes with possibly multiple boundaries the proposed method minimizes a novel surface energy. The energy prefers surfaces that follow the primitives such that the completed parts are effectively closed by extended primitives. However, in case no suitable primitives exists the method automatically and gracefully resorts to a completion by minimal surfaces.

6.1 Discussion

The algorithms developed in this work show that primitive shapes can indeed be helpful for efficient processing of large point-clouds. This is because primitive shapes possess several advantageous assets which can be directly exploited. First of all, due to the primitives' comparatively small number of parameters they can be efficiently and robustly detected. Because of the simple nature of the primitives, the user controllable settings for the detection algorithm are very intuitive and are directly related to easily understandable geometric properties such as distance and normal deviation which moreover are often of immediate relevance to the ultimate application. Despite this simplicity, the primitives can nonetheless often give a very close approximation of the input point-cloud. This way a single primitive can subsume a large number of points and serve as a basis for an efficient encoding of the small deviations to recover the original input geometry. At the same time, detected primitives are likely to correlate with semantically relevant structures. Therefore primitives lend themselves to recognition tasks as well as simple editing operations such as copy-and-paste. Finally, the primitives can be extended into regions where no points are available in order to complete the geometry. Since primitives provide a close approximation of the input data, the extension will usually be reasonable and also close to the true but unobserved surface parts.

However, there are also several drawbacks inherent to primitive-based point-cloud processing. The most obvious and arguably also most severe restriction is that primitives can usually only provide useful information on man-made scenes. But since a considerable portion of all point-cloud acquisition takes place in man-made environments, specialized solutions for these scenarios are not only justified but indeed actively called for. From a more technical point of view, another limitation is that primitives do not offer a full-scale multi-resolution representation as would for instance be desirable for compression and/or rendering. While primitives can be detected at different scales by allowing greater deviations in both euclidian distance as well as normal directions, the resulting segmentations do not in general lend themselves to hierarchical representations in a straightforward manner. Moreover, the greater the allowed tolerances, the more another drawback of the proposed detection technique becomes noticeable: The detection operates in a greedy manner and optimality of the decomposition is not guaranteed. For comparatively small tolerances, as used throughout this work, the suboptimality of the detection algorithm is mostly irrelevant because there is only a very limited set of possible segmentations since they have to fit the data tightly. Once the allowed tolerances are increased though, the number of possible segmentations dramatically increases and the suboptimality of the algorithm clearly shows up. Indeed, the effect is further intensified by the fact that in the RANSAC approach candidates are generated only samples of the original data and are therefore necessarily relatively closely aligned with the input surface. Finally, the types of detectable primitives are restricted. On the one hand this is because the number of parameters must be relatively small in order for the detection algorithm to remain efficient and robust, on the other hand the compression method requires simple parametrizations and the completion needs primitives that can be extended in a clearly specified manner.

6.2 Future work

While this work has been able to demonstrate the merit of primitive-based point-cloud processing there still remain many avenues to pursue for future research in this area. For detection of primitives, further improvements in efficiency can be envisioned, especially if additional information is known in advance, such as e.g. the largest possible extent of a single primitive or the relative pose of certain primitives. Also, work on the optimality of the decomposition is required which retains the efficiency of the approach. A possible approach might be to relax the greedy removal of points after detection of a primitive - retaining all points though would deteriorate runtime performance and some sensible trade-off is required.

With respect to compression of point-clouds, incremental techniques are likely to become of importance in the near future. Newest scanning technology captures several millions of points per second and incremental processing and visualization of this data is desirable for several reasons: First of all it would be very beneficial to be able to visualize the measured data even during the acquisition campaign in order to monitor the progress and quickly identify regions that have been inadequately captured and where acquisition needs to be repeated. Secondly, already existing point-clouds will be constantly amended by the addition of new measurements and it would be very inefficient to restart compression of the entire point-cloud each time. Due to the huge amount of data acquired per second, compression - while, as demonstrated in this work, already beneficial in many situations on large, but static point-clouds - will become indispensable for many incremental processing and visualization tasks in the near future. Since most of the large-scale acquisition campaigns take place in man-made environments (cities, archeological sites, industrial compounds, etc.) it can be expected that compression with the aid of detected primitives will also be effective in these scenarios. While the methods in this work can serve as a starting point for work in this direction, several challenges remain: The primitive detection has to be adapted to deal with incremental addition of new point samples, i.e. points that can be assigned to already existing primitives have to be quickly identified while new primitives have to be estimated for previously unobserved surface parts. The compression of surface details must use novel techniques that can be incrementally updated (Vector Quantization as used in this work is clearly not suitable) and are at the same time still amenable to interactive rendering.

In the context of recognition, a promising direction of future research is the automatic detection of rigid symmetries based on the detected primitives. For instance, combining the graph-matching methodology of Chapter 4 with a RANSAC-based generation of possible matches as suggested by Berner et al. [BBW+08] could be a viable approach. The use of primitives would lift the algorithm from working on a per-point level to operating on a set of surface patches. As a result, not only the runtime performance of the method should be improved but also the stability. Especially the influence of small holes and noise can be expected to be better handled by a primitive-based approach. Last but not least, a primitive based approach could potentially allow for generalized symmetries in the sense that matches no longer need to adhere to a rigid transform but could allow for stretches along zero-curvature directions of the primitives. For instance, it would be possible to match similar parts that are connected by cylinders of different length. Such a kind of symmetries is often encountered in man-made environments, in particular in industrial compounds.

Detected symmetries might also be beneficial to the completion of objects in several respects: First, symmetric parts can often be directly used to fill gaps in the point-cloud. Second, if regular patterns are also detected, the patterns can be extended into empty regions (much like the algorithm currently does with the primitives) and the current method can be used to automatically resolve ambiguities and intersections between different recurring patterns. Third, symmetries can be used to equalize the primitive-based completion of holes in similar parts of a model if no complete symmetric match should be available (as e.g. in Fig. 5.11).

Other advancements of the completion method could consider the optimization method. The currently employed graph-cuts are not efficient enough for processing of large volumes even though the banded 'Touch-Expand' approach does alleviate that restriction somewhat. Optimization techniques operating on a continuous domain might constitute an advantageous alternative. As recently demonstrated by Kolev et al. [KKBC09] energy functionals reminiscent of the one in Eq. (5.1) used in our completion approach can be solved in a globally optimal manner by variational methods and convex relaxation. Possibly, similar strategies can also be applied in our case. Since these variational methods can be solved on the GPU, impressive speed-up can be expected and larger volumes should become manageable.

Finally, further areas of point-cloud processing could benefit from the use of primitives. Possible uses could for instance be envisioned for advanced editing purposes, e.g. based on the stretchable principle described above. This could possibly be extended, in conjunction with symmetry detection, to the derivation of parametric procedural model descriptions.

BIBLIOGRAPHY

[AA03a] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In Leif Kobbelt, Peter Schröder, and Hugues Hoppe, editors, *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 230–239, Aachen, Germany, 2003. Eurographics Association.

[AA03b] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 272, Washington, DC, USA, 2003. IEEE Computer Society.

[AA04] Marc Alexa and Anders Adamson. On normals and projection operators for surfaces defined by point sets. In Markus Gross, Hanspeter Pfister, Marc Alexa, and Szymon Rusinkiewicz, editors, *Symposium on Point-Based Graphics*, pages 149–155, Zürich, Switzerland, 2004. Eurographics Association.

[AB81] E. H. Adelson and P. J. Burt. Image data compression with the laplacian pyramid. In *Pattern Recognition and Image Processing*, pages 218–223, 1981.

[ABCO$^+$01] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. *IEEE Visualization 2001*, pages 21–28, October 2001. ISBN 0-7803-7200-x.

[ABCO$^+$03] 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/March 2003.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*, 1973:420–??, 2001.

[AK04a] Nina Amenta and Yong J. Kil. The domain of a point set surface. In Markus Gross, Hanspeter Pfister, Marc Alexa, and Szymon

Rusinkiewicz, editors, *Symposium on Point-Based Graphics*, pages 139–147, Zürich, Switzerland, 2004. Eurographics Association.

[AK04b] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. *ACM Transactions on Graphics*, 23(3):264–270, August 2004.

[Aka73] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, pages 267–281, Budapest, 1973. Akademiai Kaidó.

[ARS99] K. Alsabti, S. Ranka, and V. Singh. An efficient space-partitioning based algorithm for the K-means clustering. *Lecture Notes in Computer Science*, 1574:355–359, 1999.

[BA02] Kenneth P. Burnham and David R. Anderson. *Model Selection and Multi-Model Inference: A Practical Information-Theoretic Approach.* Springer, 2002.

[BAC96] Andrew C. Beers, Maneesh Agrawala, and Navin Chaddha. Rendering from compressed textures. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 373–378, New York, NY, USA, 1996. ACM Press.

[BBW$^+$08] Alexander Berner, Martin Bokeloh, Michael Wand, Andreas Schilling, and Hans-Peter Seidel. A graph-based approach to symmetry detection. In *Symposium on Volume and Point-Based Graphics*, pages 1–8, Los Angeles, CA, 2008. Eurographics Association.

[BBW$^+$09] Martin Bokeloh, Alexander Berner, Michael Wand, Hans-Peter Seidel, and Andreas Schilling. Symmetry detection using line features. *Computer Graphics Forum (Proc. EUROGRAPHICS)*, 28(2):697–706, 2009.

[BF81] R. C. Bolles and M. A. Fischler. A ransac-based approach to model fitting and its application to finding cylinders in range data. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 637–643, 1981.

[BF05] Toby P. Breckon and Robert B. Fisher. Plausible 3D colour surface completion using non-parametric techniques. In Ralph R. Martin, Helmut E. Bez, and Malcolm A. Sabin, editors, *IMA Conference on the Mathematics of Surfaces*, volume 3604 of *Lecture Notes in Computer Science*, pages 102–120. Springer, 2005.

[BFR98] Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *KDD*, pages 9–15, 1998.

[BG85] C. D. Bei and R. M. Gray. An improvement of the minimum distortion encoding algorithm for vector quantization. *IEEE Transactions on Communications*, 33:1132–1133, 1985.

[BGRS99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.

[BGV$^+$02] P Benko, K. Géza, T. Várady, L. Andor, and R. Martin. Constrained fitting in reverse engineering. *Comput. Aided Geom. Des.*, 19(3):173–205, 2002.

[BHG06] Alireza Bab-Hadiashar and Niloofar Gheissari. Range image segmentation using surface selection criterion. *IEEE Transactions on Image Processing*, 15(7):2006–2018, 2006.

[BHGS06] Tamy Boubekeur, Wolfgang Heidrich, Xavier Granier, and Christophe Schlick. Volume-surface trees. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2006)*, 25(3):399–406, 2006.

[BJ88] P. J. Besl and R. C. Jain. Segmentation through variable-order surface fitting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(2):167–192, 1988.

[BK03a] Mario Botsch and Leif Kobbelt. High-quality point-based rendering on modern gpus. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 335, Washington, DC, USA, 2003. IEEE Computer Society.

[BK03b] Yuri Boykov and Vladimir Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *ICCV*, pages 26–33, 2003.

[Bli82] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics*, 16:21–29, 1982.

[BMV01] Pál Benkö, Ralph R. Martin, and Tamás Várady. Algorithms for reverse engineering boundary representation models. *Computer-Aided Design*, 33(11):839–851, 2001.

[BNdB99] Gunilla Borgefors, Ingela Nyström, and Gabriella Sanniti di Baja. Computing skeletons in three dimensions. *Pattern Recognition*, 32(7):1225–1236, 1999.

[BPK05] Stephan Bischoff, Darko Pavic, and Leif Kobbelt. Automatic restoration of polygon models. *ACM Trans. Graph.*, 24(4):1332–1352, 2005.

[BSCB00] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *SIGGRAPH*, pages 417–424, 2000.

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[BWK02] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 53–64, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[Cap05] D. P. Capel. An effective bail-out test for RANSAC consensus scoring. In *Proc. British Machine Vision Conf.*, pages 629–638, 2005.

[CDD$^+$04] Ulrich Clarenz, Udo Diewald, G. Dziuk, Martin Rumpf, and R. Rusu. A finite element method for surface restoration with smooth boundary conditions. *CAGD*, pages 427–445, 2004.

[CDF92] A. Cohen, I. Daubechies, and J. C. Feauveau. Biorthogonal bases for compactly supported wavelets. *Comm. Pure & Applied Math*, 45:485–560, 1992.

[CG01] T. Chaperon and F. Goulette. Extracting cylinders in full 3-d data using a random sampling method and the gaussian image. In *VMV01*, pages 35–42, 2001.

[CHP$^+$79] C. Csuri, R. Hackathorn, R. Parent, W. Carlson, and M. Howard. Towards an interactive high visual complexity animation system. volume 13, pages 289–299, August 1979.

[Chr07] Per H. Christensen. *Point-Based Graphics*, chapter 8.4 Point Clouds and Brick Maps for Movie Production. Morgan Kaufmann Publishers, May 2007.

[CL96] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312, 1996.

[CLF02] Umberto Castellani, Salvatore Livatino, and Robert B. Fisher. Improving environment modelling by edge occlusion surface completion. *3DPVT*, pages 672–675, 2002.

[CM05] Ondřej Chum and Jiří Matas. Matching with PROSAC - progressive sample consensus. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 220–226, Los Alamitos, USA, June 2005. IEEE Computer Society.

[CM08] Ondrej Chum and Jiri Matas. Optimal randomized RANSAC. *IEEE Trans. Pattern Anal. Mach. Intell*, 30(8):1472–1482, 2008.

[CMK03] Ondřej Chum, Jiří Matas, and Josef Kittler. Locally optimized RANSAC. In J. van Leeuwen G. Goos, J. Hartmanis, editor, *DAGM 2003: Proceedings of the 25th DAGM Symposium*, number 2781 in LNCS, pages 236–243, Heidelberger Platz 3, 14197, Berlin, Germany, September 2003. Springer-Verlag.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM Press.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, 2004.

[CSM05] Nicu D. Cornea, Deborah Silver, and Patrick Min. Curve-skeleton applications. In *IEEE Visualization*, page 13. IEEE Computer Society, 2005.

[DCOY03] Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. Fragment-based image completion. *ACM Trans. Graph.*, 22(3):303–312, 2003.

[DDSD03] Xavier Décoret, Frédo Durand, François Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *Proceedings of the ACM Siggraph*. ACM Press, 2003.

[DG04] Tamal K. Dey and Samrat Goswami. Provable surface reconstruction from noisy samples. In Jack Snoeyink and Jean-Daniel Boissonnat, editors, *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA*, pages 330–339. ACM, 2004.

[DGS05] T. K. Dey, S. Goswami, and J. Sun. Extremal surface based projections converge and reconstruct with isotopy. *Technical Report OSU-CISRC-4-05-TR25*, april 2005.

[DLS05] Tamal K. Dey, Gang Li, and Jian Sun. Normal estimation for point clouds: A comparison study for a voronoi based method. In Marc Alexa, Szymon Rusinkiewicz, Mark Pauly, and Matthias Zwicker, editors, *Symposium on Point-Based Graphics*, pages 39–46, Stony Brook, NY, 2005. Eurographics Association.

[DMGL02] J. Davis, S. R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In *3DPVT*, pages 428–861, 2002.

[DS05] Tamal K. Dey and Jian Sun. An adaptive MLS surface for reconstruction with guarantees. In Mathieu Desbrun and Helmut Pottmann, editors, *EG Symposium on Geometry Processing*, pages 43–52, Vienna, Austria, 2005. Eurographics Association.

[Elk03] Charles Elkan. Using the triangle inequality to accelerate k-means. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 147–153. AAAI Press, 2003.

[EM03] M. El-Mehalawi. A database system of mechanical components based on geometric and topological similarity. part ii: indexing, retrieval, matching, and similarity assessment. *Computer-Aided Design*, 35(1):95–105, January 2003.

[EMA03] Mohamed El-Mehalawi and Allen. A database system of mechanical components based on geometric and topological similarity. part i: representation. *Computer-Aided Design*, 35(1):83–94, January 2003.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[FCOS05] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T. Silva. Robust moving least-squares fitting with sharp features. *ACM Transactions on Graphics*, 24(3):544–552, July 2005.

[FEF97] Andrew W. Fitzgibbon, David W. Eggert, and Robert B. Fisher. High-level CAD model acquisition from range images. *Computer-aided Design*, 29(4):321–330, 1997.

[FS06a] Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 135–143, New York, NY, USA, 2006. ACM.

[FS06b] T. Funkhouser and P. Shilane. Partial matching of 3d shapes with priority-driven search. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 131–142, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[GBS03] Paulo F. U. Gotardo, Olga R. P. Bellon, and Luciano Silva. Range image segmentation by surface extraction using an improved robust estimator. *cvpr*, 02:33, 2003.

[GCO06] Ran Gal and Daniel Cohen-Or. Salient geometric features for partial shape matching and similarity. *ACM Transactions on Graphics*, 25(1):130–150, 2006.

[GD98] J. P. Grossman and William J. Dally. Point sample rendering. In George Drettakis and Nelson L. Max, editors, *Rendering Techniques*, pages 181–192. Springer, 1998.

[GG92] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic, Boston, 1992.

[GG04] Natasha Gelfand and Leonidas J. Guibas. Shape segmentation using local slippage analysis. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 214–223, New York, NY, USA, 2004. ACM Press.

[GG07] Gaël Guennebaud and Markus Gross. Algebraic point set surfaces. *ACM Transactions on Graphics*, 26(3):23:1–23:??, July 2007.

[GGG08] Gaël Guennebaud, Marcel Germann, and Markus H. Gross. Dynamic sampling and rendering of algebraic point set surfaces. *Comput. Graph. Forum*, 27(2):653–662, 2008.

[GKIS05] Stefan Gumhold, Zachi Karni, Martin Isenburg, and Hans-Peter Seidel. Predictive point-cloud compression. In *Sketch in Visual Proceedings of ACM SIGGRAPH*, 2005.

[GM97] Gideon Guy and Gérard G. Medioni. Inference of surfaces, 3D curves, and junctions from sparse, noisy, 3D data. *IEEE Trans. Pattern Anal. Mach. Intell*, 19(11):1265–1277, 1997.

[GRS98] Guha, Rastogi, and Shim. CURE: An efficient clustering algorithm for large databases. *SIGMODREC: ACM SIGMOD Record*, 27, 1998.

[GSH$^+$07] Ran Gal, Ariel Shamir, Tal Hassner, Mark Pauly, and Daniel Cohen-Or. Surface reconstruction using local shape priors. In *Eurographics SGP*, pages 253–262, 2007.

[GTK92] A. Gruss, S. Tada, and Takeo Kanade. A vlsi smart sensor for fast range imaging. In *Proc. IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 349 – 358, July 1992.

[HB89] Berthold Klaus Paul Horn and Michael J. Brooks. *Shape from Shading*. MIT Press, Cambridge, Massachusetts, 1989.

[HDD$^+$92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, 1992. ACM Press.

[HDD$^+$93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH*, volume 27, pages 19–26, 1993.

[HDD$^+$94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In *SIGGRAPH*, pages 295–302. ACM, 1994.

[HH88] H.-M. Hang and B.G. Haskell. Interpolative vector quantization of color images. *IEEE Transactions on Communications*, 36:465–470, April 1988.

[Hir08] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Trans. Pattern Anal. Mach. Intell*, 30(2):328–341, 2008.

[HK06] Alexander Hornung and Leif Kobbelt. Robust reconstruction of watertight 3D models from non-uniformly sampled point clouds without normal information. In *Eurographics SGP*, pages 41–50, 2006.

[HMHB06] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. The quantized kd-tree: Efficient ray tracing of compressed point clouds. In *IEEE Symposium on Interactive Ray Tracing*, pages 105–113, September 2006.

[HMHB07] Erik Hubo, Tom Mertens, Tom Haber, and Philippe Bekaert. Self-Similarity-Based Compression of Point Clouds, with Application to Ray Tracing. In *Eurographics Symposium on Point-Based Graphics*, pages 129–137. Eurographics Association, 2007.

[HOP$^+$05] M. Hofer, B. Odehnal, H. Pottmann, T. Steiner, and J. Wallner. 3d shape recognition and reconstruction based on line element geometry. In *Tenth IEEE International Conference on Computer Vision*, volume 2, pages 1532–1538. IEEE Computer Society, 2005.

[Hou62] P.V.C. Hough. Method and means for recognizing complex patterns. In *US Patent*, 1962.

[HPKG06] Y. Huang, J. Peng, C.-C. Jay Kuo, and M. Gopi. Octree-based progressive geometry coding of point clouds. In M. Botsch and B. Chen, editors, *Symposium on Point-Based Graphics 2006*. Eurographics, July 2006.

[HSKK01] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proceedings of ACM SIGGRAPH*, 2001.

[HY01] Mark H. Hansen and Bin Yu. Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454):746–774, 2001.

[HZ04] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, June 2004.

[IK87] J. Illingworth and J.V. Kittler. The adaptive hough transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5):690–698, September 1987.

[IK88] J. Illingworth and J. Kittler. A survey of the hough transform. *Comput. Vision Graph. Image Process.*, 44(1):87–116, 1988.

[IK01] D. Ivanov and Y. Kuzmin. Spatial patches - a primitive for 3d model representation. *Computer Graphics Forum*, 20:511–521(11), September 2001.

[JB97] Xiaoyi Jiang and Horst Bunke. Range image segmentation: Adaptive grouping of edges into regions. In *ACCV '98: Proceedings of the Third Asian Conference on Computer Vision-Volume II*, pages 299–306, London, UK, 1997. Springer-Verlag.

[JC95] Henrik Wann Jensen and Niels Jorgen Christensen. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics*, 19(2):215–224, 1995.

[JKS08] P. Jenke, B. Krückeberg, and W. StraSSer. Surface reconstruction from fitted shape primitives. In *13th International Fall Workshop Vision, Modeling and Visualization*, pages 31–40. Akademische Verlagsgesellschaft Aka GmbH, Heidelberg, oct 2008.

[JT04] Jiaya Jia and Chi-Keung Tang. Inference of segmented color and texture description by tensor voting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):771–786, 2004.

[Ju04] Tao Ju. Robust repair of polygonal models. *ACM Trans. Graph.*, 23(3):888–895, 2004.

[JWB$^+$06] Philipp Jenke, Michael Wand, Martin Bokeloh, Andreas Schilling, and Wolfgang Straßer. Bayesian point cloud reconstruction. *Comput. Graph. Forum*, 25(3):379–388, 2006.

[KB05] Vladimir Kolmogorov and Yuri Boykov. What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. In *ICCV*, pages 564–571, 2005.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Eurographics SGP*, pages 61–70, 2006.

[KBSS01] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In *SIGGRAPH*, pages 57–66, 2001.

[Kei06] Richard Keiser. *Meshless Lagrangian Methods for Physics-Based Animations of Solids and Fluids*. PhD thesis, ETH Zürich, 2006.

[KGKB03] George Kollios, Dimitrios Gunopulos, Nick Koudas, and Stefan Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Trans. Knowl. Data Eng*, 15(5):1170–1187, 2003.

[KK05] H. Kawata and T. Kanai. Direct point rendering on GPU. In *Advances in Visual Computing*, pages 587–594, 2005.

[KKBC09] K. Kolev, M. Klodt, T. Brox, and D. Cremers. Continuous global optimization in multiview 3d reconstruction. *International Journal of Computer Vision*, 84(1):80–96, August 2009.

[KMN$^+$02] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[Kol05] Ravikrishna Kolluri. Provably good moving least squares. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 213, New York, NY, USA, 2005. ACM.

[Kom06] Nikos Komodakis. Image completion using global optimization. In *CVPR*, pages 442–452. IEEE Computer Society, 2006.

[KR93] J.-K. Kim and S.-W. Ra. A fast mean-distance-ordered partial codebook search algorithm for image vector quantization. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, 40:576–579, 1993.

[KSW05] Jens Krüger, Jens Schneider, and Rüdiger Westermann. Duodecim - a structure for point scan compression and rendering. In *Proceedings of the Symposium on Point-Based Graphics 2005*, 2005.

[KV05] Aravind Kalaiah and Amitabh Varshney. Statistical geometry representation for efficient transmission and rendering. *ACM Trans. Graph.*, 24(2):348–373, 2005.

[KZ04] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Trans. on Pattern Anal. and Mach. Intell.*, 26(2):147–159, 2004.

[LB87] I. J. Leontaritis and S. A. Billings. Model selection and validation methods for nonlinear systems. *Int. J. Control*, 45(1):311–341, 1987.

[LB07] Victor S. Lempitsky and Yuri Boykov. Global optimization for shape fitting. In *CVPR*, pages 1–8, 2007.

[LBG80] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Trans. on Communications*, COM-28(1):84–95, January 1980.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH*, 21(4):163–169, 1987.

[Lev03] David Levin. Mesh-independent surface interpolation. *Geometric Modeling for Scientific Visualization*, pages 37–49, 2003.

[LGB95] Ale&#353; Leonardis, Alok Gupta, and Ruzena Bajcsy. Segmentation of range images as the search for geometric parametric models. *Int. J. Comput. Vision*, 14(3):253–277, 1995.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, New York, NY, USA, 1996. ACM Press.

[Lie03] Peter Liepa. Filling holes in meshes. In *Eurographics SGP*, pages 200–205, 2003.

[LJI$^+$03] K. Lou, S. Janyanti, N. Iyer, Y. Kalyanaraman, S. Prabhakar, and K. Ramani. A reconfigurable 3d engineering shape search system part ii: database indexing, retrieval and clusturing. In *DETC*, 2003.

[LJS97] Aleš Leonardis, Aleš Jaklič, and Franc Solina. Superquadrics for segmenting and modeling range data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(11):1289–1295, 1997.

[LPC$^+$00] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[LSSK09] Bao Li, Ruwen Schnabel, Jin Shiyao, and Reinhard Klein. Variational surface approximation and model selection. *Computer*

Graphics Forum (Proc. of Pacific Graphics), 28(7):1985–1994, October 2009.

[MC02] Jiri Matas and Ondrej Chum. Randomized RANSAC with t(d, d) test. In *Proceedings of the British Machine Vision Conference 2002 (BMVC)*, 2002.

[MC05] Jiri Matas and Ondrej Chum. Randomized RANSAC with sequential probability ratio test. In *ICCV*, pages 1727–1732. IEEE Computer Society, 2005.

[MGGP06] N. J. Mitra, L. Guibas, J. Giesen, and M. Pauly. Probabilistic fingerprints for shapes. In *Symposium on Geometry Processing*, pages 121–130, 2006.

[MGP06] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph.*, 25(3):560–568, 2006.

[MLM01] David Marshall, Gabor Lukacs, and Ralph Martin. Robust segmentation of primitives from range data in the presence of geometric degeneracy. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(3):304–314, 2001.

[MLT00] G. Medioni, M. S. Lee, and C. K. Tang. *A Computational Framework for Segmentation and Grouping*. Elsevier, 2000.

[MMG06] Bruce Merry, Patrick Marais, and James Gain. Compression of dense and regular point clouds. *Computer Graphics Forum*, 25(4):709–716, 2006.

[MNG04] 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, pages 261–276, 2004.

[MPSR01] David McWherter, Mitchell Peabody, Ali C. Shokoufandeh, and William Regli. Database techniques for archival of solid models. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 78–87, New York, NY, USA, 2001. ACM Press.

[MSM04] Timothy Mattson, Beverly Sanders, and Berna Massingill. *Patterns for Parallel Programming*. Addison-Wesley Longman, Amsterdam, September 2004.

[MTN+02] D.R. Myatt, P.H.S. Torr, S.J. Nasuto, J.M. Bishop, and R. Craddock. Napsac: High noise, high dimensional robust estimation - it's in the bag. In *BMVC*, page Computer Vision Tools, 2002.

[NB77] Ramakant Nevatia and Thomas O. Binford. Description and recognition of curved objects. *Artificial Intelligence*, 8(1):77–98, 1977.

[Nis05] David Nistér. Preemptive RANSAC for live structure and motion estimation. *Mach. Vision Appl.*, 16(5):321–329, 2005.

[ÖGG09] C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2):493–501, 2009.

[OH06] Tilo Ochotta and Stefan Hiller. Hardware rendering of 3d geometry with elevation maps. In *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*, page 10, Washington, DC, USA, 2006. IEEE Computer Society.

[OLY05] Ratko Orlandic, Ying Lai, and Wai Gen Yee. Clustering high-dimensional data using an efficient and effective data space reduction. In Otthein Herzog, Hans-Jörg Schek, Norbert Fuhr, Abdur Chowdhury, and Wilfried Teiken, editors, *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 201–208. ACM, 2005.

[OS04] Tilo Ochotta and Dietmar Saupe. Compression of point-based 3d models by shape-adaptive wavelet coding of multi-height fields. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 103–112, June 2004.

[OS08] Tilo Ochotta and Dietmar Saupe. Image-based surface compression. *Comput. Graph. Forum*, 27(6):1647–1663, 2008.

[Pau03] Mark Pauly. *Point Primitives for Interactive Modeling and Processing of 3D Geometry*. PhD thesis, ETH Zürich, 2003.

[PBJB98] Mark W. Powell, Kevin W. Bowyer, Xiaoyi Jiang, and Horst Bunke. Comparing curved-surface range image segmenters. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 286, Washington, DC, USA, 1998. IEEE Computer Society.

[PG01] Mark Pauly and Markus Gross. Spectral processing of point-sampled geometry. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 379–386, New York, NY, USA, 2001. ACM Press.

[PGK02] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In Robert J. Moorhead, Markus Gross, and Kenneth I. Joy, editors, *Proceedings of IEEE Visualization 2002*, pages 163–170. IEEE Computer Society, IEEE Computer Society Press, October 2002.

[PH98] J. S. Pan and K. C. Huang. A new vector quantization image coding algorithm based on the extension of the bound for minkowski metric. *Pattern Recognition*, 31(11):1757–1760, 1998.

[PLH$^+$05] Helmut Pottmann, Stefan Leopoldseder, Michael Hofer, Tibor Steiner, and Wenping Wang. Industrial geometry: recent advances and applications in CAD. *Computer-Aided Design*, 37(7):751–766, 2005.

[PM99] Dan Pelleg and Andrew W. Moore. Accelerating exact k -means algorithms with geometric reasoning. In *KDD*, pages 277–281, 1999.

[PMG$^+$05] Mark Pauly, Niloy J. Mitra, Joachim Giesen, Markus Gross, and Leonidas J. Guibas. Example-Based 3D Scan Completion. In *Eurographics SGP*, pages 23–32, 2005.

[PMW$^+$08] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. Guibas. Discovering structural regularity in 3D geometry. *ACM Trans. on Graph.*, 27(3):1–11, 2008.

[PR05] Joshua Podolak and Szymon Rusinkiewicz. Atomic Volumes for Mesh Completion. In *Eurographics SGP*, pages 33–41, 2005.

[PSBM07] Valerio Pascucci, Giorgio Scorzelli, Peer-Timo Bremer, and Ajith Mascarenhas. Robust on-line computation of reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007.

[Ree83] William T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Trans. Graph*, 2(2):91–108, 1983.

[Ree85] W. Reeves. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *Proceedings of SIGGRAPH'85*, page 313, 1985.

[RFP08] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus. In David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman, editors, *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part II*, volume 5303 of *Lecture Notes in Computer Science*, pages 500–513. Springer, 2008.

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[RL79] Jorma J. Rissanen and Glen G. Langdon, Jr. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, March 1979.

[RL93] Gerhard Roth and Martin D. Levine. Extracting geometric primitives. *CVGIP: Image Underst.*, 58(1):1–22, 1993.

[RL00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[RSA08] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. *ACM Trans. Graph.*, 27(3):1–9, 2008.

[SACO04] Andrei Sharf, Marc Alexa, and Daniel Cohen-Or. Context-based surface completion. *ACM Trans. on Graph.*, 23(3):878–887, August 2004.

[SB95] Nickolas S. Sapidis and Paul J. Besl. Direct construction of polynomial surfaces from dense range images through region growing. *ACM Trans. Graph.*, 14(2):171–200, 1995.

[Sch78] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.

[SDF01] Freek Stulp, Fabio Dell'Acqua, and Robert Fisher. Reconstruction of surfaces behind occlusions in range images. In *3D Digital Imaging and Modeling*, pages 232–239, 2001.

[SDK09] Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and reconstruction with primitive shapes. *Computer Graphics Forum (Proc. of Eurographics)*, 28(2):503–512, March 2009.

[SF06] Philip Shilane and Thomas Funkhouser. Selecting distinctive 3D shape descriptors for similarity retrieval. In *Shape Modeling International*, June 2006.

[SF07] Philip Shilane and Thomas Funkhouser. Distinctive regions of 3D surfaces. *ACM Transactions on Graphics*, 26(2):Article 7, June 2007.

[Sha98] Craig M. Shakarji. Least-squares fitting algorithms of the NIST algorithm testing system. *J. Res. Nat. Inst. Stand. Techn.*, 103(6):633–641, 1998.

[SK06] R. Schnabel and R. Klein. Octree-based point-cloud compression. In M. Botsch and B. Chen, editors, *Symposium on Point-Based Graphics 2006*. Eurographics, July 2006.

[SLSK07] Andrei Sharf, Thomas Lewiner, Ariel Shamir, and Leif Kobbelt. On–the–fly curve-skeleton computation for 3d shapes. In *Eurographics*, pages 323–328, Prague, september 2007.

[SMK07] Ruwen Schnabel, Sebastian Möser, and Reinhard Klein. A parallelly decodeable compression scheme for efficient point-cloud rendering. In *Symposium on Point-Based Graphics 2007*, pages 214–226, September 2007.

[SMK08] Ruwen Schnabel, Sebastian Möser, and Reinhard Klein. Fast vector quantization for efficient rendering of compressed point-clouds. *Computers and Graphics*, 32(2):246–259, April 2008.

[SSGD03] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson. Skeleton based shape matching and retrieval, 2003.

[Ste97] C. V. Stewart. Bias in robust estimation caused by discontinuities and multiple structures. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(8):818–833, August 1997.

[SW03] Jens Schneider and Rudiger Westermann. Compression domain volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 39, Washington, DC, USA, 2003. IEEE Computer Society.

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[SWWK08] Ruwen Schnabel, Raoul Wessel, Roland Wahl, and Reinhard Klein. Shape recognition in 3d point-clouds. In V. Skala, editor, *The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008*. UNION Agency-Science Press, February 2008.

[SZS$^+$08] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(6):1068–1080, 2008.

[TL07] Gary K. L. Tam and Rynson W. H. Lau. Deformable model retrieval based on topological and geometric signatures. *IEEE TVCG*, 13(3):470–482, 2007.

[TM02] Ben Tordoff and David W. Murray. Guided sampling and consensus for motion estimation. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *Computer Vision - ECCV 2002, 7th European Conference on Computer Vision, Copenhagen, Denmark, May 28-31, 2002, Proceedings, Part I*, volume 2350 of *Lecture Notes in Computer Science*, pages 82–98. Springer, 2002.

[TV04] J. W. Tangelder and R. C. Veltkamp. A survey of content based 3d shape retrieval methods. In *SMI '04: Proceedings of the Shape Modeling International 2004 (SMI'04)*, pages 145–156, Washington, DC, USA, 2004. IEEE Computer Society.

[TZ98] Phil Torr and Andrew Zisserman. Robust computation and parametrization of multiple view relations. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 727, Washington, DC, USA, 1998. IEEE Computer Society.

[TZ00] P. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78:138–156, 2000.

[VCBS03] Joan Verdera, Vicent Caselles, Marcelo Bertalmío, and Guillermo Sapiro. Inpainting surface holes. In *ICIP (2)*, pages 903–906, 2003.

[VGSR04] G. Vosselman, B.G.H. Gorte, G. Sithole, and T. Rabbani. Recognising structure in laser scanner point clouds. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 46(8/W2):33–38, 2004.

[VKH06] Vivek Verma, Rakesh Kumar, and Stephen Hsu. 3d building detection and modeling from aerial lidar data. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2213–2220, Washington, DC, USA, 2006. IEEE Computer Society.

[VMC97] Tamás Várady, Ralph R. Martin, and Jordan Cox. Reverse engineering of geometric models - an introduction. *Computer-Aided Design*, 29(4):255–268, 1997.

[Vos02] George Vosselman. Fusion of laser scanning data, maps, and aerial photographs for building reconstruction. In *Geoscience and Remote Sensing Symposium*. IEEE International, 2002.

[Wal47] A. Wald. *Sequential Analysis*. Dover, 1947.

[WB68] C. S. Wallace and D. M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.

[WGE$^+$04] M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray, and S. Würmlin. Progressive compression of point-sampled models. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 95–102, June 2004.

[WGK05] R. Wahl, M. Guthe, and R. Klein. Identifying planes in point-clouds for efficient hybrid rendering. In *The 13th Pacific Conference on Computer Graphics and Applications*, October 2005.

[WK04] Jainhua Wu and Leif Kobbelt. Optimized sub-sampling of point sets for surface splatting. *Computer Graphics Forum*, 23(3):643–652, 2004.

[WK05] J. Wu and L. Kobbelt. Structure recovery via hybrid variational surface approximation. In *Computer Graphics Forum*, volume 24, pages 277–284, 2005.

[WNC87] I. Witten, R. Neal, and J. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30:520–541, 1987.

[WO02] Jianning Wang and Manuel M. Oliveira. Improved scene reconstruction from range images. *Computer Graphics Forum*, 21(3):521–530, September 2002.

[XO93] Lei Xu and Erkki Oja. Randomized hough transform (RHT): basic mechanisms, algorithms, and computational complexities. *CVGIP: Image Underst.*, 57(2):131–154, 1993.

[You98] Abdou Youssef. Parallel algorithms for entropy-coding techniques. *NISTIR (NISTIR 6113)*, December 1998.

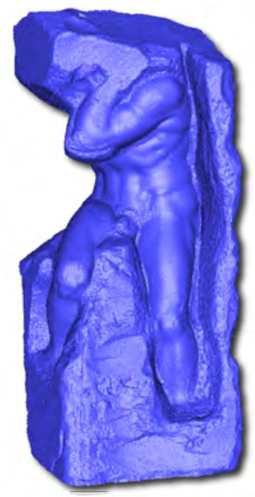
[ZPKG02] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: an interactive system for point-based surface editing. *ACM Transactions on Graphics*, 21(3):322–329, July 2002.

[ZPvBG01] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, New York, NY, USA, 2001. ACM Press.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 103–114, New York, NY, USA, 1996. ACM.

[ZTS02] Emanoil Zuckerberger, Ayellet Tal, and Shymon Shlafman. Polyhedral surface decomposition with applications. *Computers and Graphics*, 26(5):733–743, October 2002.

DATA SOURCES



Atlas, Digital Michelangelo Project, Stanford University, USA, http://graphics.stanford.edu/projects/mich/

Box, Holger Wirth, Metronom Automation GmbH, http://www.metronom-automation.de

Carter, model is provided courtesy of INRIA and ISTI by the AIM@SHAPE Shape Repository, http://shapes.aimatshape.net/

David, Digital Michelangelo Project, Stanford University, USA, http://graphics.stanford.edu/projects/mich/

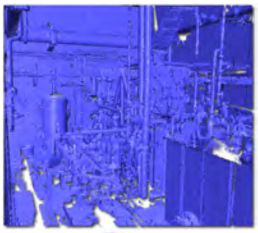
Dragon, 3D Scanning Repository, Stanford University, USA, http://graphics.stanford.edu/data/3Dscanrep/

Ephesos, Michael Wimmer, TU Vienna, http://www.cg.tuwien.ac.at/staff/MichaelWimmer.html

Fandisk, Hugues Hoppe, Microsoft Research, ftp://ftp.research.microsoft.com/users/hhoppe/data/thesis/input_pts/

FemaleWB and MaleWB, Cyberware Inc., USA, www.cyberware.com

Graz, courtesy of Heiko Hirschmüller, German Aerospace Center (DLR) – Institute of Robotics and Mechatronics. Model was derived by semi-global matching from Vexcel Imaging Graz$^{\text{TM}}$ imagery.



Industrial compounds, scannTec GmbH & CO. KG, http://scanntec.com

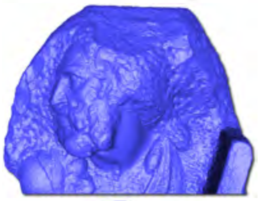
Master cylinder, model is provided courtesy of INRIA and ISTI by the AIM@SHAPE Shape Repository, http://shapes.aimatshape.net/

Oil-pump, model is provided courtesy of INRIA and ISTI by the AIM@SHAPE Shape Repository, http://shapes.aimatshape.net/

Rocker arm, Cyberware Inc., USA, www.cyberware.com

Rolling stage, model is provided courtesy of INRIA and ISTI by the AIM@SHAPE Shape Repository, http://shapes.aimatshape.net/

Santa, Cyberware Inc., USA, www.cyberware.com



St. Matthew, Digital Michelangelo Project, Stanford University, USA, http://graphics.stanford.edu/projects/mich/

Venus, Cyberware Inc., USA, www.cyberware.com

PUBLICATIONS

- Gerhard H. Bendels, Ruwen Schnabel, and Reinhard Klein. Detail-Preserving Surface Inpainting. In *Pproceedings of The 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST)*, Eurographics Association, pages 41-48, Eurographics Association, Nov. 2005

- Gerhard H. Bendels, Ruwen Schnabel, and Reinhard Klein. Detecting Holes in Point Set Surfaces. In *Journal of WSCG*, Feb. 2006, 14, pages 89-96

- Ruwen Schnabel and Reinhard Klein. Octree-based Point-Cloud Compression. In *Proceedings of Symposium on Point-Based Graphics 2006*, pp. 111-120, Eurographics, July 2006

- Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for Point-Cloud Shape Detection. In *Computer Graphics Forum*, June 2007, 26:2(214-226)

- Ruwen Schnabel, Sebastian Möser, and Reinhard Klein. A Parallelly Decodeable Compression Scheme for Efficient Point-Cloud Rendering. In *Proceedings of Symposium on Point-Based Graphics 2007*, pages 214-226, Sept. 2007

- Ruwen Schnabel, Raoul Wessel, Roland Wahl, and Reinhard Klein. Shape Recognition in 3D Point-Clouds. In *Proceedings of The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008*, UNION Agency-Science Press, Feb. 2008

- Ruwen Schnabel, Roland Wahl, and Reinhard Klein. RANSAC Based Out-of-Core Point-Cloud Shape Detection for City-Modeling. In *Schriftenreihe des DVW, Terrestrisches Laser-Scanning (TLS 2007)*, 2007

- Ruwen Schnabel, Sebastian Möser, and Reinhard Klein. Fast Vector Quantization for Efficient Rendering of Compressed Point-Clouds. In *Computers and Graphics*, (Apr. 2008), 32:2(246-259)

- Patrick Degener, Ruwen Schnabel, Christopher Schwartz, and Reinhard Klein. Effective Visualization of Short Routes. In *IEEE Transactions on Visualization and Computer Graphics*, Oct. 2008, 14:6(1452-1458)
- Fabian Giesen, Ruwen Schnabel, and Reinhard Klein. Augmented Compression for Server-Side Rendering. In *Proceedings of Vision, Modeling, and Visualization 2008 (VMV 2008)*, Akademische Verlagsgesellschaft Aka GmbH, Heidelberg, Oct. 2008
- Roland Wahl, Ruwen Schnabel, and Reinhard Klein. From Detailed Digital Surface Models to City Models Using Constrained Simplification, In *Photogrammetrie, Fernerkundung, Geoinformation (2008)*:3(207-215)
- Ruwen Schnabel, Patrick Degener, and Reinhard Klein. Completion and Reconstruction with Primitive Shapes. In *Computer Graphics Forum (Proc. of Eurographics)*, Mar. 2009, 28:2(503-512)
- Bao Li, Ruwen Schnabel, Jin Shiyao, and Reinhard Klein. Variational Surface Approximation and Model Selection. In *Computer Graphics Forum (Proc. of Pacific Graphics)*, Oct. 2009, 28:7(1985-1994)
- Christopher Schwartz, Ruwen Schnabel, Patrick Degener und Reinhard Klein. PhotoPath: Single Image Path Depictions from Multiple Photographs. In: *Journal of WSCG*, Feb. 2010, 18:1-3
- Bao Li, Ruwen Schnabel, Reinhard Klein, Zhiquan Cheng, Gang Dang und Jin Shiyao. Robust normal estimation for point clouds with sharp features. In: *Computers and Graphics*, Apr. 2010, 34:2(94-106)
- Roland Ruiters, Ruwen Schnabel und Reinhard Klein. Patch-based Texture Interpolation. In *Computer Graphics Forum*, June 2010, 29:4(1421-1429)