Kinetic data structures for the geometric modeling of urban environments

Jean-Philippe Bauchet

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Structures de données cinétiques pour la modélisation géométrique d'environnements urbains

Kinetic data structures
for the geometric modeling of urban environments

Jean-Philippe BAUCHET

LuxCarta Techonology – INRIA

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur

Dirigée par : Florent Lafarge

Soutenue le : 6 décembre 2019

Devant le jury, composé de :

Guillaume Damiand, rapporteur, Université Claude Bernard
Yasutaka Furukawa, rapporteur, Simon Fraser University
Pierre Alliez, examinateur, INRIA – Université Côte d'Azur
George Drettakis, examinateur, INRIA – Université Côte d'Azur
Florent Lafarge, examinateur, INRIA – Université Côte d'Azur
Yuliya Tarabalka, examinatrice, LuxCarta Technology

LuxCarta

Inria

Acknowledgements

First of all, I wish to thank to my advisor, Florent, for his support and hard-criticism towards my work, which encouraged me to give the best of myself. I spent three tough but awesome years at Inria, and it has been a sincere pleasure to work with you.

Naturally, I would like to thank Lionel Laurore, Justin Hyland and Albéric Maumy, as cofounders of LuxCarta, for making this experience possible. Thank you for your trust. You left me free to explore my own research paths, with the help of my advisor, and I truly appreciated it. As a research engineer at LuxCarta, I am now looking forward to working on new enthralling topics that will contribute to the success of the company.

Many thanks, also, to the committee members, for the time spent reading this thesis, for your insightful comments and constructive remarks.

Now, I wish to thank the numerous friends I have got at Inria, who have been part of this long journey. I have been so glad to work in such an international atmosphere, with amazing people coming from all horizons. I had a great time at Inria, but also outside office, and I keep our summer trip to Denmark as one of my greatest memories. Special thanks to Timothée for the countless Duck Game sessions during lunchbreak, but also to Nicolas and Gaëtan for bringing card games at work !

Finally, I would like to express my gratitude towards my family for their unfailing support. You are always describing me as a source of pride, which touches me a lot, and I hope to stay worthy of it. Last but not least, thank you Flora, for your constant kindness, your smile and your moral support, which all have been very precious to me, especially during the final months of this thesis. Thank you !

Abstract

The geometric modeling of urban objects from physical measurements, and their representation in an accurate, compact and efficient way, is an enduring problem in computer vision and computer graphics. In the literature, the geometric data structures at the interface between input physical measurements and output models typically suffer from scalability issues, and fail to partition 2D and 3D bounding domains of complex scenes.

In this thesis, we propose a new family of geometric data structures that relies on a kinetic framework. More precisely, we compute partitions of bounding domains by detecting geometric shapes such as line-segments and planes, and extending these shapes until they collide with each other. This process results in light partitions, containing a low number of polygonal cells. We propose two geometric modeling pipelines, one for the vectorization of regions of interest in images, another for the reconstruction of concise polygonal meshes from point clouds. Both approaches exploit kinetic data structures to decompose efficiently either a 2D image domain or a 3D bounding domain into cells. Then, we extract objects from the partitions by optimizing a binary labeling of the cells.

Conducted on a wide range of data in terms of contents, complexity, sizes and acquisition characteristics, our experiments demonstrate the scalability and the versatility of our methods. We show the applicative potential of our method by applying our kinetic formulation to the problem of urban modeling from remote sensing data.

Keywords: Kinetic data structures, image partitioning, object contouring, surface reconstruction, surface approximation, polygonal surface mesh, energy minimization, 3D modeling, urban scene reconstruction, Lidar data

Résumé

La modélisation géométrique d'objets urbains à partir de mesures physiques et leur représentation de manière efficace, compacte et précise est un problème difficile en vision par ordinateur et en infographie. Dans la littérature scientifique, les structures de données géométriques à l'interface entre les mesures physiques en entrée et les modèles produits en sortie passent rarement à l'échelle et ne permettent pas de partitionner des domaines fermés 2D et 3D représentant des scènes complexes.

Dans cette thèse, on étudie une nouvelle famille de structures de données géométrique qui repose sur une formulation cinétique. Plus précisément, on réalise une partition de domaines fermés en détectant et en propageant au cours du temps des formes géométriques telles que des segments de droites ou des plans, jusqu'à collision et création de cellules polygonales. On propose en particulier deux méthodes de modélisation géométrique, une pour la vectorisation de régions d'intérêt dans des images, et une autre pour la reconstruction d'objets en maillages polygonaux concis à partir de nuages de points 3D. Les deux approches exploitent les structures de données cinétiques pour décomposer efficacement en cellules soit un domaine image en 2D, soit un domaine fermé en 3D. Les objets sont ensuite extraits de la partition à l'aide d'une procédure d'étiquetage binaire des cellules.

Les expériences menées sur une grande variété de données en termes de nature, contenus, complexité, taille et caractéristiques d'acquisition démontrent la polyvalence de ces deux méthodes. On montre en particulier leur potentiel applicatif sur le problème de modélisation urbaine à grande échelle à partir de données aériennes et satellitaires

Mots clés: structures de données cinétiques, partitionnement d'images, reconstruction de surfaces, extraction d'objets, approximation de surfaces, maillage polygonaux surfaciques, minimisation d'énergies, modélisation 3D, reconstruction de scènes urbaines, données Lidar

Contents

Page

Contents **ix**

1 Introduction **1**
- 1.1 Context and motivations . . . 1
- 1.2 Data sources . . . 2
  - 1.2.1 Optical imagery . . . 2
    - 1.2.1.1 General description . . . 2
    - 1.2.1.2 Aerial imagery . . . 2
    - 1.2.1.3 Satellite imagery . . . 3
  - 1.2.2 Point clouds . . . 4
    - 1.2.2.1 Lidar data . . . 4
    - 1.2.2.2 Multi-View Stereo data . . . 5
    - 1.2.2.3 RGB-D cameras . . . 7
- 1.3 Challenges . . . 8
- 1.4 Outline . . . 9

2 Related works **11**
- 2.1 Object contouring by polygonal shapes . . . 11
  - 2.1.1 Direct extraction of polygons . . . 11
  - 2.1.2 Vectorization pipelines . . . 12
  - 2.1.3 Polygonal partitions . . . 14
    - 2.1.3.1 Line-segment detection . . . 15
    - 2.1.3.2 Image partitioning . . . 16
- 2.2 Generation of concise polygonal meshes from point clouds . . . 17
  - 2.2.1 Approximation methods . . . 17
  - 2.2.2 Simplification methods . . . 18
  - 2.2.3 Shape assembling methods . . . 19
    - 2.2.3.1 Planar shape detection . . . 19
    - 2.2.3.2 Connectivity-based methods . . . 21
    - 2.2.3.3 Slicing-based methods . . . 21
    - 2.2.3.4 Data structures for space partitioning . . . 22

3 Our contributions **25**
- 3.1 Limitations of previous works . . . 25
- 3.2 Contributions . . . 26
  - 3.2.1 Kinetic framework for partitioning images and 3D spaces 27
  - 3.2.2 Global regularization of 2D and 3D primitives . . . 28

3.2.3 Algorithm of 2D and 3D object extraction 28
3.2.4 Benchmark for concise mesh generation 28
3.2.5 City reconstruction pipeline 29

4 Shape detection **31**
4.1 Line-segment detection and regularization 31
4.1.1 Choice of a line-segment detector 31
4.1.2 Regularization of a set of 2D line-segments 31
4.2 Plane detection and regularization 35
4.2.1 Choice of a plane detection technique 35
4.2.2 Regularization of a set of 3D planar primitives 35
4.2.2.1 Re-orienting the planes 36
4.2.2.2 Re-aligning the planes 41
4.3 Conclusions 44

5 Polygonal partitioning in 2D **45**
5.1 Background on kinetic data structures 45
5.2 Algorithm 46
5.3 Experiments 50
5.3.1 Control on the complexity of the partition 50
5.3.2 Comparison with superpixel methods 50
5.3.3 Results on aerial and satellite imagery 54
5.4 Limitations 58
5.5 Conclusion 58

6 Polyhedral partitioning in 3D **59**
6.1 Algorithm 59
6.2 Implementation details 65
6.3 Experiments 68
6.3.1 Kinetic vs. exhaustive partitions 68
6.3.2 Spatial subdivision 68
6.4 Conclusion 70

7 Object extraction **73**
7.1 Object contouring 73
7.1.1 Description of the model 73
7.1.2 Experiments and comparisons 74
7.1.3 Limitations 75
7.2 Surface extraction 76
7.2.1 Description of the model 76
7.2.2 Experiments 78
7.2.2.1 Flexibility 79

7.2.2.2 Robustness to imperfect data . . . 83
7.2.2.3 Performances . . . 84
7.2.2.4 Impact of spatial subdivision . . . 86
7.2.3 Comparisons . . . 87
7.2.3.1 Ablation study . . . 87
7.2.3.2 Surface reconstruction methods . . . 90
7.2.3.3 Surface approximation pipelines . . . 94
7.2.4 Limitations . . . 95
7.3 Conclusions . . . 106

8 Application to urban modeling **107**
8.1 Related works . . . 107
8.2 Pipeline . . . 109
8.2.1 Input . . . 109
8.2.2 Classification . . . 109
8.2.3 Building contouring . . . 111
8.2.4 Building reconstruction . . . 111
8.3 Experiments . . . 113
8.4 Conclusions . . . 118

9 Conclusion and perspectives **119**
9.1 Summary . . . 119
9.2 Perspectives . . . 121

A Appendix **127**
A.1 Polyhedral partitioning in 3D: a pseudo-code . . . 127

Bibliography **131**

Chapter 1

Introduction

1.1 Context and motivations

The geometric modeling of urban objects and environments from physical measurements is a long-discussed topic within the computer vision, computer graphics and remote sensing communities. It has a wide range of practical applications, which mainly lie on three aspects: manufacturing, rendering and environmental simulations.

Recent years have witnessed a technological evolution of the sensors used to acquire data. As a result, the collected datasets get much larger. This emphasizes the need for scalable algorithms, capable of processing massive amounts of data. However, the complexity of the models produced by geometric modeling techniques is also of critical importance for some applications. For instance, simulating the propagation of acoustic or electromagnetic waves in urban areas requires accurate, as well as simple models. Computer-aided designed models can be used to perform physical simulations, but the creation of such models is too time-consuming and labor-intensive. For large and complex scenes, automated tools should be preferred.

This thesis addresses the problem of converting data issued from physical measurements into geometric models with the five following objectives:

1. Fidelity. Our models should constitute faithful approximations of the observed data.

2. Simplicity. These models should be composed of a low number of geometric elements, ideally just enough, given a user-defined tolerance approximation error.

3. Efficiency. We are interested in designing algorithms that are fast, scalable, and proceed with a low memory consumption.

4. Automation. These algorithms should be fully automatic and only take into account a few intuitive and user-defined parameters.

5. Genericity. Our algorithms should be applicable on all types of scenes and objects, without relying on any particular geometric assumption.

1.2 Data sources

Various data types may be considered for the geometric modeling of urban objects or scenes. In this thesis, we will focus on two usual data sources: optical imagery and 3D point clouds.

1.2.1 Optical imagery

1.2.1.1 General description

Digital cameras are now accessible to a general public. A camera is built upon an image sensor, that converts captured light into an analogic electric signal. This signal is then amplified and digitalized to obtain an image: a bidimensional array of red, green and blue integer values. An entry of this table is called a pixel.

Nowadays, standard cameras have an image resolution exceeding 10 millions of pixels. Compact cameras are sold at low prices, and are also embedded into most smartphones. Users collect and share images on the Internet using applications such as Flickr, Instagram, or Google Maps. Since the contents of these images often depict urban objects, Web databases may constitute a valuable data source for geometric modeling and urban reconstruction.

1.2.1.2 Aerial imagery

Aerial imagery is a popular technique to acquire data for the geometric modeling of large urban scenes. It consists in taking a digital camera aboard a hot-air balloon, a drone or a plane that flies over an area, following a predefined flight map. The further processing of the images by photogrammetry tools, for instance to generate digital elevation models, may require overlapping constraints between the images.

We distinguish oblique and vertical aerial imagery. In oblique imagery, photographs are taken at an angle relative to the Earth's surface. The no-

tions of low, or high oblique imagery refer to the value of this angle. In contrast, vertical photographs are taken from a straight down angle.

Aerial images can be combined for various purposes, including image stitching for the creation of panoramas, and 3D reconstruction. Vertical images, for their part, can be used to generate orthophotos, which are simulations of photographs taken at infinite distance, without perspective effects. Orthophotos have applications in GIS systems, and urban planning offices may be interested in extracting all building footprints, or road networks, from such images.

One of the contraints posed by aerial imagery can be its high acquisition cost. Also, depending on countries, the use of aircrafts may be restricted by local authorities.

1.2.1.3 Satellite imagery

Satellite imagery is a commonly used data source for natural and Earth sciences, such as meteorology, oceanography or forestry. It was initially restricted to military applications, and observation purposes. However, the development of commercial satellite imagery has attracted an increasing attention from the computer vision community, and the technological advances in terms of image resolution now open the door to precise cartography of urban environments.

Launched in 2014 by DitigalGlobe, WorldView-3 is a satellite that has, for instance, a panchromatic resolution of 0.31 meters per pixel. WorldView-3 also has a multispectral and infrared resolution at the scale of the meter. In addition, it has an average revisit time of less than once a day, and can collect up to 680 000 km^2 per day.

For companies and land surveying offices, one of the main strengths of satellite imagery comes from its significantly lower acquisition cost compared to aerial imagery. On the other hand, acquisition depends on atmospheric conditions and requires a weak cloud cover. Furthermore, the automatic processing of these massive amounts of data raises a concern for the scalability of the applied methods.

1.2.2 Point clouds

1.2.2.1 Lidar data

General description. Standing for "LIght Detection And Ranging", Lidar [Cra07] is a surveying method that measures distance to a target. This technology was initially used for meteorological applications. However, it is nowadays commonly used for creating high-resolution topographic maps in land surveying, as well as for navigation and control in autonomous transportation systems.

A Lidar scanner is first composed of a laser emitter, that generates luminous beams directed towards an object. Then, the reflected light pattern is captured by a photodetector and converted to an electric impulse, which is further analyzed by an embedded signal processing chain. According to the time-of-flight principle, the time difference between the emission of the signal and the reception of its most important echo is proportional to the distance between the scanner and the object.

The wavelengths of beams emitted by Lidar scanners vary between 250 nanometers to 10 micrometers. Depending on the application context, beams of different wavelengths can be emitted by Lidar devices. The choice of an appropriate wavelength is indeed motivated by the absorbtion and backscattering properties of the target material. Airborne topographic mapping Lidar systems operate in the near-infrared domain with a typical wavelength of 1064 nanometers, when bathymetric systems rather use a wavelength of 532 nanometers to better penetrate waterbodies.

Better target resolution is achieved with shorter pulses, and Lidar scanners can emit up to several hundreds of thousands of pulses per second. This enables very dense and precise measurements of target objects, with a sampling error at the scale of the centimeter or even the millimeter, depending on the acquisition mode (aerial or terrestrial). For this reason, Lidar scans constitute a valuable data source for the geometric modeling or urban objects or environments. However, the very high cost of laser scanners curbs the use of this technology.

Although Lidar scanners were primarily designed to measure distances, extra information are sometimes provided by these devices for each captured point, such as the amplitude of the signal, or the number of echoes. This information depends on the target material, and can be used for classification

purposes. For instance, the number of echoes is a discriminative feature for detecting vegetation in airborne Lidar data.

Terrestrial Lidar scans. Terrestrial Lidar scans are a way to acquire Lidar data. In a stationary mode, the scanner is usually mounted on a tripod. Due to occlusions, multiple scans at different locations can be required to obtain a full dataset, for instance when scanning an indoor scene. The fusion step can be performed using the iterative closest point algorithm [BM92]. However, we can observe the absence of measured data in areas that are too close to the sensor.

A laser scanner can also be mounted on a vehicle, generally for ground-based urban reconstruction purposes. For example, Zhao *et al* [ZS01] describe an acquisition protocol in which two one-dimensional lasers scan facades of buildings along the horizontal and vertical axes. Backpack systems can also be used for acquiring high resolution data in complex indoor environments [LCC+10].

For terrestrial Lidar scans, the density of points is usually comprised between 100 and 3000 points per squared meter, with a sampling error at the scale of the millimeter.

Airborne Lidar scans. Airborne Lidar scans can be obtained by mounting a scanner on a plane or an unmanned aerial vehicle. This enables the capture of data coverage of large areas like cities, in a short amount of time. In urban modeling, one of the main difficulties posed by airborne Lidar data comes from the fact that the sensor is oriented downwards. As a consequence, vertical structures such as facades are often completely or almost completely missing from the generated datasets. We might also observe missing holes over reflective surfaces, such as waterbodies.

The density of points clouds in airborne Lidar scans is lower than in terrestrial scans, since it varies between 1 and 50 points per square meter. Not surprisingly, the precision also decreases at the scale of the centimeter, even the decimeter.

1.2.2.2 Multi-View Stereo data

Recovering 3D geometry of an object from photographs is a major research topic in computer vision. Given several images of the same object or scene,

Multi-View Stereo (MVS) vision techniques refer to the estimation of the most likely 3D shape that explains the observed photographs, under known assumptions of materials, viewpoints, and lighting conditions, using stereo correspondances as their main cue [FH+15]. Some early techniques showed the potential of large crowd-sourced dabatases of images for 3D reconstruction [SSS06].

Typical MVS pipelines consist of two steps. Over a first phase, the camera parameters are estimated for each image. Then, the 3D geometry is reconstructed from the image and the set of camera parameters.

The first step is achieved using Structure from Motion (SfM) techniques. Based on the pinhole camera model, SfM techniques output the camera parameters of each image (focal length and projection matrix), but also a sparse set of 3D locations that are associated to subsets of pixels in the input images. This is done by a feature extraction step in images [Low04], a feature matching step in which a set of candidate pairs of overlapping images is identified, followed by a geometric verification step in which a transformation is estimated to map feature points across images. Examples of SfM techniques are described in [AFS+11, HSDF15, SF16].

Then, a dense representation of the scene is generated by MVS techniques. According to the review of Seitz *et al* [SCD+06], MVS algorithms can be categorized into four classes: voxel-based methods [SMP07], surface evolution based methods [HKLP09], patch-based methods [FP09] and depth map based methods [SZFP16, XT19]. Within the scope this thesis, we have been led to use some results generated by the algorithm of Schönberger *et al* [SZFP16]. In a nutshell, this method jointly optimizes depth and normal information in a large collection of images using priors strengthening the photometric and geometric consistency of the result.

As illustrated by the Tanks and Temples benchmark [KPZK17], MVS techniques now achieve high-quality 3D reconstructions of large indoor or outdoor scenes from collections of images, with an error at the scale of the centimeter. Algorithms such as Poisson reconstruction [KBH06] can be used to turn the obtained point clouds into smooth surfaces. The latter are however complex and cannot be used in many applications. Still, the point clouds resulting from MVS techniques provide a good basis for the geometric modeling of concise meshes.

1.2.2.3 RGB-D cameras

Recent years have witnessed the rise of mass-manufactured, affordable and hand-held RGB-D cameras. Microsoft started the development of the Kinect in 2010. Initially designed as an entertainement device, it associates a video stream recorded by a RGB camera to a depth sensor. The depth is acquired by stuctured light: a pattern is projected into the scene using infrared (IR) light. The deformed pattern is recorded by an IR camera and the depth is calculated from the deformation.

Computer graphics and computer vision researchers quickly understood the potential of this affordable technology in a context of 3D reconstruction [NIH+11, IKH+11]. RGB-D sensors can be integrated to robots for the mapping of unknown environments, which is of great interest for military operations or emergency situations. This problem is known as Simultaneous Mapping And Localization (SLAM) and is a major topic in robotics and computer vision. There exists a vast literature on the broader topic of 3D reconstruction of static or dynamic scenes using RGB-D cameras, recently surveyed by [ZSG+18].

The reconstruction of dynamic scenes is beyond the scope of this thesis. We refer to the aforementioned survey for more details. In order to reconstruct a static object or scenes from a RGB-D stream, a typical pipeline takes the shape of a loop incorporating the following steps. Firstly, the depth map is preprocessed and denoised, using for instance a bilateral filter. Secondly, camera poses are estimated. Various strategies can be explored, including frame-to-frame tracking, frame-to-model tracking that locates the camera with respect to the so-far reconstructed model, or global pose estimation. Finally, the 3D geometry of the scene is obtained. Again, several representation schemes can be used: a voxel grid associated to an implicit distance function, or a sparse set of point locations that can be later converted into a smooth mesh.

3D reconstruction from RGB-D sensors now achieves a excellent level of detail, with an average reconstruction error of a few millimeters, including for large scenes. However the meshes generated by these techniques are generally complex and require to be simplified.

1.3 Challenges

The geometric modeling of urban enviroments is a difficult problem that meets three main challenges: acquisition constraints, full automation, and quality of output models [MWA+13].

Acquisition constraints. We observe a set of recurrent defects in data captured from the real world, that geometric modeling techniques should be able to handle.

Noise, for instance, is a typical problem. It can directly result from the sensor, or be caused by failures in the data registration process. Outliers are also frequent, and are the consequence of the presence of unwanted, yet unavoidable objects in the scene. We may think of temporary objects such as pedestrians, traffic signs or cars in a context of urban modeling. Likewise, it is difficult to deal with heterogeneous sampling conditions. The density of points acquired by a terrestrial laser scanner decreases with its distance to the captured objects in the scene. On the other hand, overlapped areas in aerial LiDAR scans are described by more points than other areas, whereas facades are often partly or completely missing in these scans, due to geometric constraints. Also, some materials like glass cannot be captured by laser scanners. Lighting conditions can also play a role in the quality of the obtained data, and shadows may pose specific problems to some image processing algorithms. Finally, the absence of data covering specific parts of an object, due to occlusions, also constitutes a challenge in computer vision.

Full automation. The ultimate goal of geometric modeling is to provide efficient and fully automatic algorithms.

Interactive modeling techniques can be proposed for some specific application fields, like in architecture or in the entertainement industry where the expected quality of the models is beyond the capacities of the start-of-the-art techniques. However, interactive techniques are inadapted to the processing of large amounts of data, like in remote sensing where industrials and researchers aim at parsing the Earth's entire surface. Therefore, there is not only a quest for accurate, but also for fast and scalable algorithms. However, one of the major difficulties in the design of such techniques comes from the diversity or urban objects and landscapes. Even within a small scene, very different objects can be found in terms of shape and appearance.

Some geometric modeling techniques rely on assumptions, characterizing for example the regularity of the objects to extract, but these approaches fail to model entire scenes. On the other hand, assumption-free algorithms are more flexible, but tend to produce less structured results.

Quality of models. Depending on the application context, different criteria can be used to assess the quality of models generated by geometric modeling techniques.

Geometric accuracy, i.e. the quality of fit to the measured data, is the most straightforward way to evaluate a model. In some situations, some geometric guarantees reinforcing the regularity of the result can also be expected. The compactness of the model can also be a very important criterion, enabling further uses for reverse engineering or physical simulations, for instance. On the other hand, the entertainement industry lays a bigger emphasis on the visual aspect of the generated model.

We observe that these evaluation criteria may conflict with each other. For instance, a less complex model can be obtained at the price of a higher geometric error. The search for a unified metric combining these criteria remains an open problem in geometric modeling, where ground truth data is not always available.

1.4 Outline

In the next chapter, we review the problem of object extraction and approximation from images and point clouds. We discuss the limitations of these methods, and present our contributions in chapter 3.

Chapters 4 to 7 present our pipeline for object extraction from images or point clouds. In a nutshell, our algorithm first detects a set of primitives that may be regularized (chapter 4). These primitives are then used to create a partition of the appropriate space (chapters 5 and 6). Finally, we formulate the object extraction step as a labelling problem of the polygonal or polyhedral cells, depending on the dimension (chapter 7).

In chapter 8, we show the applicative potential of this pipeline by addressing the specific problem of city modeling from airborne LiDAR data. We finally draw conclusions of our works in chapter 9.

Chapter 2

Related works

Numerous scientific challenges are related to image analysis and understanding. In this chapter, we review the problem of object vectorization in images. In contrast to traditional segmentation methods, which approximate different objects of interest in an image as multiple regions of pixels, free-form polygons offer a compact and structure-aware representation of these objects. Polygons are particularly adapted to the extraction of man-made objects, such as rooftop buildings in aerial images, which are characterized by a few number of vertices and strong geometric signatures. We present various strategies that focus on this difficult problem.

This chapter also focuses on the generation of concise polygonal meshes from point clouds. This is a distinct problem from dense mesh generation, which focuses on reconstructing a smooth representation from a sampled shape. Here, we review strategies that approximate a 3D object using a small number of meaningful facets.

2.1 Object contouring by polygonal shapes

In our review of algorithms addressing the problem of object contouring by polygonal shapes, we distinguish three main types of methods : direct approaches, vectorization pipelines and methods based on polygonal partitions.

2.1.1 Direct extraction of polygons

A first category of works focuses on the direct extraction of polygonal shapes from images. A possible strategy consists in detecting a set of geometric shapes from the image, such as line-segments, before assembling them to obtain the closed contour of an object. Sun *et al* [SCF14] for instance, address this problem by identifying plausible elementary cycles in adjacency graphs

of line-segments. In contrast, Zhang *et al* [ZCS+10, ZFW+12] generate additional line-segments to connect the previously detected line fragments, and find an optimal cycle by applying the ratio-contour algorithm [WKSW05].

Stochastic approaches coupled with simulated annealing might also be used to detect parametric shapes in images, such as rectangles [KvLS07, LDZPD08]. However, these methods are harmed by a slow convergence speed.

Some recent works focus on learning the geometric characteristics of polygons. Polygon-RNN [CKUF17, ALKF18], for instance, is a semi-automatic object annotation tool via polygons. Assuming the existence of a bounding box enclosing an object instance, this methods sequentially predicts the vertices of a polygon tightening the object using a recurrent neural network (RNN). Although efficient, these mechanisms offer no shape control on the final polygons, which may be approximated by a large number of points. Moreover, since the RNN only keeps in memory the two last predicted vertices, polygons generated by this technique may contain self-intersections. The latter issue has been addressed in a recent work [LGK+19] in which all vertices of a bounding polygon or curve are simultaneously predicted, using a graph convolutional network.

In a similar vein, PolyCNN extracts building rooftops from remote sensing images [GT18], but these polygons are restricted to quadrilaterals. PolyMapper [LWL18] alleviates this problem and directly predicts vectorial city maps that include road networks and complex rooftop shapes. However, these approaches remain heavily data-dependent, and the output polygons come up with no geometric guarantees, such as orthogonality.

2.1.2 Vectorization pipelines

Vectorization pipelines are composed of two steps. Over a first phase, an object is extracted from an input image as a region of pixels. Then, a line approximation algorithm is used to simplify its contours and obtain a polygon.

There exists several approaches to extract a given object from an image. One may refer, for instance, to interactive image segmentation methods such as Intelligent Scissors [MB98] or GrabCut [RKB04].

Another strategy consists in over-segmenting the image before extracting

Figure 2.1: Vectorization vs. direct extraction of polygons. Left: a pixel-wise segmentation of aerial images, predicted by a neural network [LQQ+18]. Vectorizing the contours of each building instance will result in numerous approximations. Right: polygons are directly extracted using an adapted architecture [LWL18]. Image taken from [LWL18].

an object as a group of superpixels, *i.e.* perceptually homogeneous atomic regions. Decomposing an image into superpixels is a well-known problem in computer vision. Various models have been proposed [ASS+12, AS17, LTRC11, VdBBR+12, TLJ+18] in order to meet a certain set of expected properties, including adherence of the superpixels to the boundaries of visible objects. In this context, we are interested in finding a subset of superpixels whose boundaries overlap the true contours of the object, which can be formulated as a cost minimization problem [LSD10] or by designing a hierarchical segmentation algorithm [RS13].

Another popular way to extract the contours of an object consists in computing a saliency map. In a highly influential work, Cheng *et al* [CMH+14] achieve this goal by reasoning on the notion of color contrast. Learning methods, based on convolutional neural networks, can also be employed [WWL+16]. However, since convolutional neural networks operate at the scale of image patches instead of pixels, the resulting saliency maps might be blurry, especially near the boundaries of the object of interest, leading to imprecise detection. Some strategies have been recently suggested to alleviate this problem [LY16, QZH+19].

Once objects of interest are extracted, the subsequent approximation step is usually performed using the well-known Douglas-Peucker algorithm and its variants [DP73, WM03]. Alternative algorithms can be used, casting for instance shape simplification as an optimal transport problem [dGCSAD11] or by lattice refinement [TMAT18]. In a remote sensing context [ST10], the

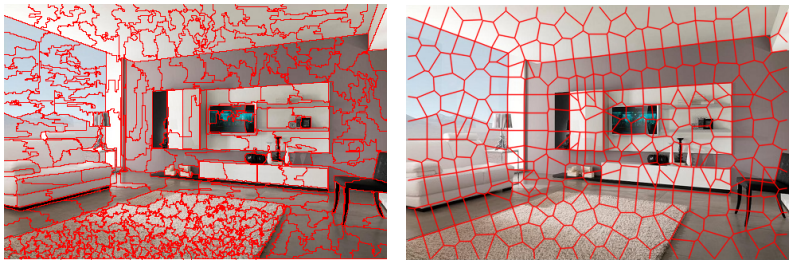


Figure 2.2: Superpixels and polygons. Left: superpixel decomposition generated by Liu *et al* [LTRC11] (300 cells). Right: polygonal image partitioning returned by Duan *et al* [DL15] (322 cells). While a polygonal cell is delimited by a few vertices, contours of superpixels must be defined as lists of pixels.

Hough transform can also be applied to an edge map to vectorize patches in raster format that correspond to regions of interest: the detected buildings in an image.

2.1.3 Polygonal partitions

Inspired by superpixel grouping approaches, another strategy for object vectorization consists in generating a partition of polygonal cells of an image, and selecting a subset of connected cells that approximate well the contours of all objects of interest.

Polygonal partitions offer some advantages over superpixel decomposition techniques. Indeed, they offer resolution-independent representations of images, contrary to superpixels which typically operate at pixel level. Polygonal partitions also benefit from a low storage cost, since the construction of polygonal cells requires only a few vertices. In contrast, borders of free-form superpixels are defined as sequences of pixels. Moreover, polygonal partitions can exploit the geometric information present in the image, and we observe that several methods build a polygonal partition using a pre-detected set of line-segments that roughly approximate the contours of the objects.

A natural way to obtain a partition of polygons in an image is to vectorize a set of superpixels. This approach is followed by Achanta *et al* [AS17], who position vertices where three superpixels meet, and simplify the sequence of pixels found in between by applying the Douglas-Peucker algorithm. However, this operation may generate approximation errors and requires a good

superpixel connectivity, which is difficult to guarantee in practice. We note that this method also returns non-convex polygons.

2.1.3.1 Line-segment detection

Fitting geometric shapes to images is an efficient way to synthesize a large number of pixels into a few parametric functions. Line-segments are the most common geometric shapes for analyzing images, especially when the observed structures are lineic. The detection of line-segments is a long-discussed topic in computer vision. Most existing algorithms are composed of two-step pipelines, in which line-segments are fitted to a previously generated heat map. These techniques may be classified into two categories.

A popular approach consists in first estimating an edge map, by means of different filters like the Canny edge detector [Can87]. Then, the Hough transform [DH71, DHH11] detects a set of prominent lines in the edge map. Therefore, extracting line-segments means identifying peaks of maximal intensity along these lines in the image. Hough-based methods offer the advantage to accumulate information from the whole image for detecting lines, but the determination of appropriate endpoinds for the line-segments is a difficult problem. Various peak-localization algorithms are discussed in the literature, for example based on the notion of persistance [KM19], on probabilistic models [ATQE17] or by analyzing the voting distribution in the Hough space [XSK14].

On the other hand, perceptual grouping approaches use image gradient as a low-level cue and locally group pixels into a set of line-segment candidates, from which false positives are later discarded. The most representative example of this family of techniques is the Line-Segment Detector (LSD) of van Gioi *et al* [vGJMR10]. This linear-time algorithm is based on the Helmholtz principle [DMM07], according to which an observed geometric structure is perceptually ε-meaningful if its expected number of occurrences is less than ε under the *a contrario* random assumption. Here, the tested structures are obviously the candidate line-segments, set as approximations of regions where the image gradient shows a similar orientation. ε is by default set to 1, a natural value. But this parameter can be tuned, thus allowing the user to control the number of false detections in the returned set of line-segments. Another example of perceptual grouping method is described in the work of Cho *et al* [CYL17].

Recently, learning methods have been used to address the problem of line-segment detection. For instance, Huang *et al* [HWZ+18] present deep convolutional neural networks for jointly retrieving line-segments and junctions from an image. Another interesting contribution is provided by Xue *et al* [XBW+19]. In this paper, the authors use a dual representation for line-segments maps: region-based attraction field maps. The line-segment detection problem from an input image thus becomes a region coloring problem. Attraction field maps are learnt by convolutional neural networks and are converted to line-segments. Although data-dependent, and computationally complex, these methods both achieve very promising results.

In this thesis, we do not bring any contribution to the problem of line-segment detection in images. However, we observe that line-segments returned by state-of-the-art algorithms may not preserve well geometric regularities of observed structures, as for instance line alignments in a regular layout of windows on a facade. This is due to noise, insufficient resolution of the Hough space [GA12], or approximation mechanisms when shrinking regions to line-segments [vGJMR10, XBW+19]. Global regularization can be a valuable tool to correct these inaccuracies, and reduce output complexity by removing redundant shapes. Existing regularization methods typically operate from 3D shapes, either by iterative refinements [OLA16] or by energy minimization [PCSS14].

2.1.3.2 Image partitioning

The output of line-segment detection algorithms can be used to generate a polygonal partition of the image.

For example, Duan *et al* [DL15] build a constrained Voronoi tessellation whose edges conform to these line-segments. A Poisson-disk sampling is then applied to the Voronoi cells to obtain uniformly-sized polygons. Gevers *et al* [GS97] and Forsythe *et al* [FKF16, FK17] build Delaunay triangulations, before regrouping triangles into polygons. The former operates by iteratively splitting triangles with heterogeneous radiometry, whereas the latter uses a constrained Delaunay triangulation with a minimal angle for all triangles, that conforms to pre-detected line-segments. The higher this angle, the more complex the partition [She02]. Although exploiting line-segments to guide the polygonal partitioning is computationally efficient, existing methods [DL15, FKF16, FK17] fail to properly recover the junctions of lineic

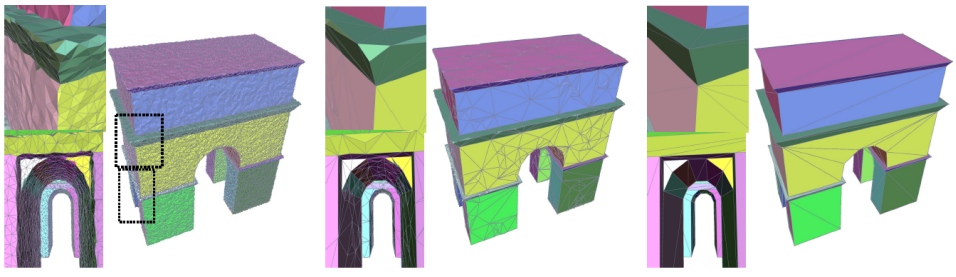
structures, leading in best situations to the generation of several polygons around a junction. Another approach [PS05] applies a constrained Delaunay triangulation of Canny edge contours, before grouping the obtained triangles based on perceptual cues.

However, let us note that the literature offers other examples of image partitioning techniques that do not necessarily rely on a set of line-segments. We mention, for instance, the work of Hermes *et al* [HB03], in which a triangular mesh is iteratively refined based on conditional entropy. Polygons are later aggregated, based on the Jensen-Shannon divergence that measures the similarity of the discrete probability distributions associated to two adjacent cells. In another recent work, Favreau *et al* propose a Delaunay point process for sampling a Delaunay triangulation in images. The triangles are later augmented with binary activation labels for object contouring purposes [FLBA19].

2.2 Generation of concise polygonal meshes from point clouds

We distinguish three families of algorithms for converting a point cloud into a concise polygonal mesh: approximation methods, simplification methods and shape assembling methods.

2.2.1 Approximation methods



(a) (b) (c)

Figure 2.3: An example of mesh approximation. (a) An input mesh, composed of 13k vertices, with a set of detected planar proxies. (b) Decimated mesh with 1300 vertices. (c) Final mesh with 115 vertices. Image taken from [SLA15].

A first way to produce concise polygonal meshes from point clouds consists in reconstructing a smooth surface from the input points before simplifying it. The first step can rely upon mature algorithms from the geometry processing toolbox, such as the popular Poisson reconstruction method [KBH06, KH13]. We may refer to a recent survey that discusses the main algorithms for smooth surface reconstruction [BTS+17].

Once extracted, the dense triangle mesh is simplified into a coarser mesh. The most common approach consists in contracting edges until reaching a target number of facets [GH97, Lin00]. To better preserve the piecewise-planar structure of objects, the edge contraction operators can account for planar shapes detected in the dense mesh [SLA15]. However, on planar parts such approaches yield triangle meshes whose adjacent facets are unlikely to be coplanar and form a meaningful decomposition. Other methods [CSAD04, CSALM13] alleviate this problem by connecting planar shapes using the adjacency inferred from the input mesh. These surface approximation methods are in general effective, but they require as input a mesh that is both geometrically and topologically accurate to deliver faithful results. Unfortunately, such as requirement is rarely guaranteed from real-world data which are often highly corrupted by noise, outliers and occlusions.

2.2.2 Simplification methods

Another line of works reduces the combinatorial problem of mesh generation by imposing geometric assumptions for the output surface. For instance, the Manhattan-World assumption [CY00] enforces the generation of polycubes [THCM04, HJS+14, IYF15] by imposing output facets to follow only three orthogonal directions. Another popular geometric assumption consists in constraining the output surface to specific disk-topologies. For instance, 2.5D view-dependent representations is typically well suited to buildings with airborne data [MWA+13] and facades with streetside images [BSRVG15]. Some approaches also approximate surfaces with specific layouts of polygonal facets. Such polyhedral patterns are particularly relevant for architectural design [JWWP14, JTV+15]. However, such geometric assumptions are only relevant for specific application domains.

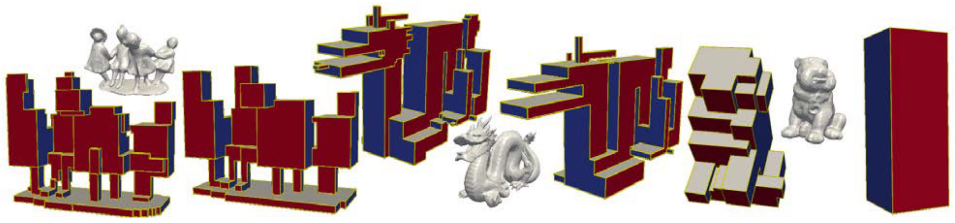


Figure 2.4: An example of mesh simplification. Tetrahedral meshes are deformed into polycube maps through minimization of the ℓ_1 norm of mesh normals. Image taken from [HJS+14].

2.2.3 Shape assembling methods

Finally, some algorithms address the problem of mesh generation by extracting a set of planar shapes from the point cloud as a preprocessing step, before assembling them into a unified polygonal mesh. Planar shapes are sets of points to which a plane is fitted, and constitute an intermediate presentation between input points and the output mesh.

We distinguish two categories of shape assembling methods, either based on connectivity graphs or plane slicing. We shall begin our study by focusing on a problem of planar shape detection in a point cloud.

2.2.3.1 Planar shape detection

The detection of geometric shapes in 3D data is a fundamental problem in computer vision and computer graphics. Many applications, including scene reconstruction or robotics, benefit from the representation of the data at an intermediate level of abstraction, by means of simple geometric shapes like planes, cuboids or cylinders. The robustness of primitive detection techniques to common defects observed in real data, such as noise, outliers, or varying sampling densities, is one of the main challenges that shape detection techniques should overcome.

In this paragraph, we review the problem of plane detection from raw and unstructured point clouds. There exists a wide range of methods addressing this problem, recently surveyed by Kaiser *et al* [KYZB18]. Each family of methods has its own advantages and drawbacks, and scores differently in terms of data fidelity, speed or scalability.

RANSAC-based methods [SWK07, SWWK08] are popular techniques of plane detection, achieving real-time performance requirements. This class of

algorithms iteratively generate random plane hypotheses from the data, and select the planes that fit the most points. However, these approaches are non-deterministic. The geometric accuracy of the final set of planes can be improved by adding a regularization procedure as a postprocessing step. Li *et al* optimize the orientation and the placement of the planes using a non-linear energetic formulation [LWC+11]. In a similar fashion, Monszpart *et al* [MMBM15] use a mixed-integer quadratic program to filter out a regular arrangement of planes from a set of candidates.

Originally used to detect sets of 2D lines in images, the Hough transform [DH71] has been extended to the detection of arbitrary shapes [Bal81]. The detection of a set of planes in a point clouds follows the same scheme as in 2D, with the search for local laxima in a discretized 3D parameter space [HSMS13, LO15, HHRB11]. This approach is particularly efficient against incomplete data, but the continuous and unbounded nature of the parameter space, and the choice of an appropriate quantization step may become a computational issue, especially when dealing with large datasets.

Region-growing algorithms [MLM01, RVDHV06] offer an interesting alternative to the strategies discussed above. These approaches consist in the propagation of a plane hypothesis from a seed point to its neighbors. The hypothesis is validated if it can be associated to a minimal number of points. The choice of appropriate seed points [OLA16] is an important algorithmic step, that may enhance running times and quality of the results. By design, region-growing approaches are slower than other methods, but they also tend to produce more accurate planes by exploiting connectivity information between points.

Shapes can also be detected by learning approaches trained from CAD model databases. In the work of Fang *et al* [FLD18], planar shapes are extracted depending on a target level of detail. Point properties can also be learnt [QYSG17] in order to estimate different types and parameters of primitives that fit the point cloud. This process is not only restricted to plane detection but may include other shapes, like cuboids or cylinders [LSD+19].

There remains some challenges in the problem of shape detection. The introduction of learning methods represents an interesting way to improve the accuracy and the semantic meaning of the returned shapes, but the enhancement of such techniques is beyond the scope of this thesis.

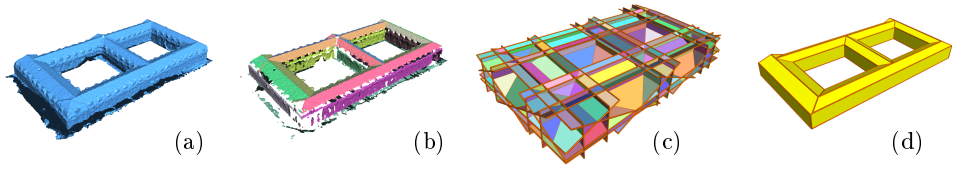


Figure 2.5: An example of shape assembling method. (a) Input point cloud. (b) Detected planar primitives. (c) Generation of a set of candidate faces, obtained through the arrangement of infinite planes supporting the planar primitives. (d) Selected faces and final mesh. Image taken from [NW17].

Let us now describe the two main families of shape assembling techniques introduced before.

2.2.3.2 Connectivity-based methods

Connectivity-based methods analyze an adjacency graph between planar shapes in order to extract the vertices, edges and facets of the output mesh [CC08, VKVLV11, SWF11]. Although these methods are fast, they are likely to produce incomplete models on challenging, defect-laden data where adjacency graphs often contain linkage errors.

One possible solution consists in completing the missing parts either interactively with user-guided mesh operations [ASF+13], or automatically with dense triangle meshes [LA13]. Unfortunately, the first alternative does not offer a relevant solution for processing large volumes of data, and the second one does not output concise polygon meshes.

2.2.3.3 Slicing-based methods

Slicing-based methods [CLP10, BDLGM14, OLA14, MMP16, NW17] are more robust to challenging data. They operate by slicing the input 3D space with the supporting planes of the detected shapes. The output partition is a decomposition into convex polyhedral cells. The output mesh is then extracted by labeling the cells as inside or outside the surface, or equivalently, by selecting the polygonal facets of the cells that are part of the surface. The main limitation is the high algorithmic complexity for constructing the 3D partition, commonly performed via a binary space partition (BSP) tree updated at each plane insertion [MF97]. Such a data structure can take hours to construct, and consume dozens of gigabytes when several hundred

shapes are considered. Moreover, the returned partitions are overly complex, comprising many small anisotropic cells which are hampering the subsequent surface extraction process. Other approaches [SDK09, VLA15] discretize the partitioning space instead of computing the exact geometry of the whole partition. This option is less costly, but often yields geometric artifacts when the discretization is not fine enough.

2.2.3.4 Data structures for space partitioning

In addition with the slicing approaches discussed before [BDLGM14, CLP10, NW17], based on BSP trees, several algorithmic data structures have been used to partition a 3D volume into polyhedra in a context of surface reconstruction.

Indeed, some algorithms rely on a voxel grid to recover an implicit surface from a point cloud. The voxel grid is seen as an occupancy map, and the surface itself is extracted using a least-square formulation [KBH06], or a graph-cut formulation [HK06, LB07], where the voxels represent the nodes of the graph. However, the computational cost of the grid can grow cubicly with the desired level of detail.

Some other methods generate a 3D Delaunay triangulation from an input point cloud. Intuitively, triangles connect points that are sampled on the same plane, and polyhedrons are elongated in a direction that is approximatively orthogonal to the inferred surface. This technique, however, requires a dense point cloud. It is also sensitive to noise. To reduce the complexity of the partition, Lafarge *et al* [LA13] propose a structuring approach. Outliers are filtered out, and the triangulation is computed from a uniformly resampled point cloud. Van Kreveld *et al* [VKVLV13], for their part, build a conformed constrained triangulation with planar polygons that approximate the point cloud.

Finally, combinatorial maps [DL14] represent an interesting alternative to partition 3D spaces into more meaningful polyhedrons. A combinatorial map is an edge-centered data structure, composed of a set of oriented half-edges called darts, and adjacency relations between these darts. Under this approach, a facet is represented by a cycle of darts, and linking adjacent facets returns a polyhedron. Starting with of soup of planar polygons describing buildings or cities, Diakite *et al* use this topological information maps to represent objects using a certain levels of detail [DDVM14] or assign

semantic labels [DDG14].

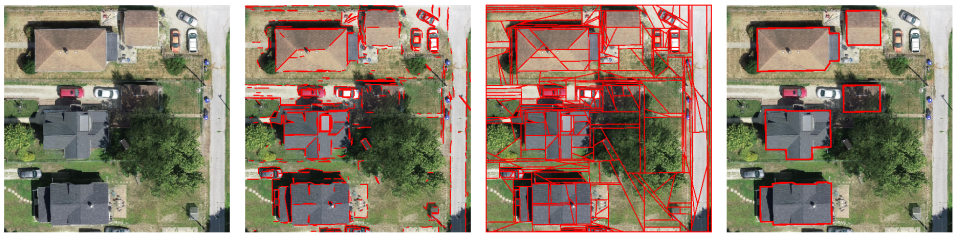
Chapter 3

Our contributions

3.1 Limitations of previous works

Our previous review shows that the vectorization of objects in images remains an open problem. Automatic direct extraction methods come at the price of slow convergence rates or require high amounts of data, and the results generated by these techniques often lack of geometric guarantees. On the other hand, vectorization pipelines seem a natural way to produce polygons, but the simplification of pixel subregions turns out to be not that easy. While simplification algorithms offer the user some control on the complexity of the output polygons, they do not take into account some structural information contained in the image, and may drift from the initial object silhouettes. As a consequence, vectorization pipelines are prone to approximation errors.

For these reasons, the construction of a polygonal partition of the image coupled with a cell activation mechanism may constitute an interesting approach in the quest for a generic vectorization tool. Most existing approaches rely on a set of pre-detected line-segments that are fitted on object contours. Although robust and offering geometric guarantees, these line-segment-based methods may lack of accuracy. Indeed, they poorly deal with potential intersections of line-segments that might occur in a spatial neighborhood. In particular, they assume line-segments overlap entirely or almost entirely the lineic components of an object. Experience shows that this is not always true, and that the line-segments need to be extended to properly capture the underlying structure. Also, Delaunay triangulations or Voronoi diagrams are quite restrictive. Finally, we observe that imposing homogeneously-sized polygons does not allow the capture of meaningful polygons as line-segments are not uniformly distributed on the image domain. Therefore, we would like to design a polygonal partition technique that addresses the problems previously exposed and brings more semantic meaning to the obtained cells.



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

Figure 3.1: Overview of our 2D vectorization pipeline. Our algorithm takes as input an image (a). We extract a set of line-segments that roughly approximate the contours of the image (b). These segments are used to generate a polygonal partition of the image, using a kinetic data structure (c). A subset of connected cells is then obtained to vectorize objects of interest in the image, here the building rooftops, with only a few vertices (d).

The generation of a concise polygonal mesh from raw input data is also a challenging task. We observe that performances of approximation methods may depend on the geometric and topological accuracy of the dense meshes they process, which is difficult to guarantee with data acquired in real conditions. Simplification methods are also powerful tools, but some geometric assumptions are in contradiction with our quest towards genericity.

Finally, the main issue of shape assembling methods is the lack of scalability of the underlying geometric data structures that produce a polyhedral decomposition of the space. Splitting the 3D space into subcells using the infinite support plane of each previously detected shape is an interesting strategy, but it is way too computationally complex and memory-intensive when several hundred planes are involved.

We now present our contributions in order to address the aforementioned issues.

3.2 Contributions

In this thesis, we present two generic, versatile and scalable pipelines for the polygonization of objects of interest in images, and the generation of concise and polygonal meshes from 3D point clouds.

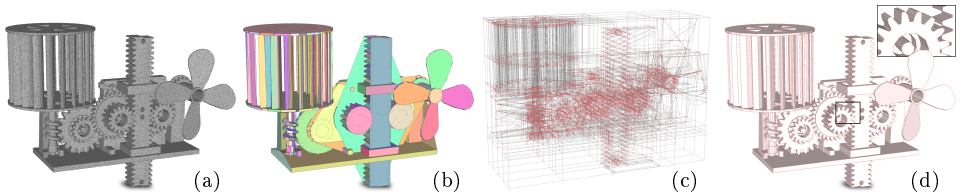


Figure 3.2: Overview of our 3D reconstruction pipeline. Our algorithm takes as input a point cloud with oriented normals (a) and a configuration of convex polygons that approximate the object (b). Convex polygons are obtained through plane detection as the convex hulls of inlier points are projected onto the plane. The 3D bounding box of the object is decomposed into polyhedra by kinetic partitioning from the convex polygons (c). A concise polygonal mesh is then extracted by labeling each polyhedron as inside or outside the object (d). Model: FULL THING.

3.2.1 Kinetic framework for partitioning images and 3D spaces

The principal contribution of this thesis is a kinetic framework for the polygonal partitioning of 2D domains, and the polyhedral partitioning of 3D spaces. In this framework, a set of pre-detected shapes grow within a certain space, using a dynamic stop criterion that allows us to control the complexity of the generated partitions.

In the 2D case, a set of line-segments progressively widen within the image, and intersect each other. We then decide whether line-segments must keep extending based on image gradient considerations, or a user-defined number of collisions. In chapter 5, we will observe that this strategy allows us to both recover better junctions in lineic structures and describe objects with polygons in a more meaningful manner than superpixel-based polygons.

In the 3D case, we consider a set of convex planar polygons detected in an input point cloud. Like for images, polygons grow inside a bounding box enclosing the point cloud until colliding each other. The stopping condition is again a fixed number of intersections or a function of the density of vertices from the point cloud near the collision point. In chapter 6 we will show that our approach yields much lighter partitions, with a significantly lower algorithmic complexity than exhaustive partitioning techniques.

3.2.2 Global regularization of 2D and 3D primitives

Our second contribution is an algorithm of global regularization of a set of line-segments.

Indeed, we observe that line-segment detection techniques do not always preserve the geometric relationships that which are initially visible in an image. In order to improve the spatial coherence of the obtained set of line-segments, Duan *et al* [DL15] proposed a solution to this problem by building an adjacency graph and locally applying a set of operations (fusion, removal, or concurrence reinforcement) to this graph. In our case, we addressed this problem at global scale, by formulating it as an energy minimization problem which aims at recovering parallelism, orthogonality and collinarity relationships between the line-segments. Our formulation is described in chapter 4 of this document.

We extend this mechanism to the regularization of a set of planar shapes, by taking into account additional relationships, including z-symmetry.

3.2.3 Algorithm of 2D and 3D object extraction

Our third contribution is a min-cut algorithm for object extraction using our kinetic partitions. Using a saliency map for images, we perform a binary labelling of the generated cells and return the closed set of edges that separate the active cells from the inactive ones. Our experiments, detailed in chapter 7, suggest that kinetic partitions may constitute a valuable tool for addressing the problem of object vectorization.

This is the same approach in 3D, where our formulation relies upon a visibility criterion that leverages the orientations of point normals to extract watertight meshes from the 3D partition. We further evaluate and compare our approach with state-of-the-art mesh generation methods, against several metrics, on a wide range of datasets in terms of complexity, size and acquisition constraints.

3.2.4 Benchmark for concise mesh generation

In this thesis, we evaluate and compare our algorithm for concise polygonal mesh generation with state-of-the-art methods against several quality and performance metrics, on a wide range of 42 datasets that differ in terms of complexity, size, and acquisition characteristics. Our results, from both

quantitative and qualitative poins of view, are presented in section 7.2.2. This evaluation material will be released online to the Computer Vision and Compter Graphics communities.

3.2.5 City reconstruction pipeline

We finally propose a computational geometry approach for addressing the problem of city reconstruction from airborne Lidar data. Our algorithm actually combines the 2D vectorization and the 3D reconstruction pipelines introduced before, and turns large datasets with millions of points into a LOD2 representation of the scene, in compliance with the CityGML format [GP12]. Under this formalism, buildings are reconstructed with piecewise-planar roofs and vertical facades. We present our method and our experiments in chapter 8.

Chapter 4

Shape detection

In this chapter, we focus on the problem of shape detection and regularization from 2D and 3D data. These shapes, also called primitives, constitute the input of the kinetic data structures that we use for modeling urban objects.

We present an algorithm of global regularization of the detected set of primitives. This algorithm reinforces geometric relationships between pairs of spatially close primitives, such as parallelism or orthogonality. We present our formulation, and show that it can be a valuable option in order to generate less complex kinetic partitions.

4.1 Line-segment detection and regularization

4.1.1 Choice of a line-segment detector

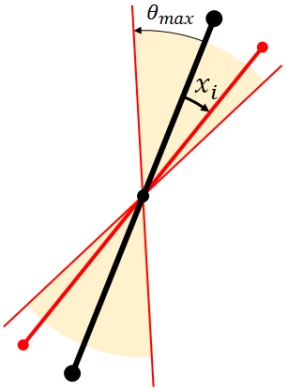
Although any of the line-segment detection techniques reviewed in chapter 2 can be used to provide an input to our vectorization pipeline, our experiments use the Line-Segment Detector (LSD) of Van Gioi *et al* [vGJMR10]. As previously mentioned, this algorithm has several interesting properties, including a linear algorithmic complexity in the number of pixels in the image, and a mostly parameterless control scheme with respect to other existing algorithms.

4.1.2 Regularization of a set of 2D line-segments

We may optionally operate a global regularization of line-segments in order to (i) correct imprecisions and (ii) reduce the occurrences of skinny cells in the subsequent image partitioning, later detailed in chapter 5. This process is mainly designed for images with man-made objects without strong perspective effects.

We propose two quadratic formulations, performed sequentially for computational efficiency, that first re-orient and then re-align line-segments with respect to the three principal geometric regularities used for characterizing

shapes of man-made objects, *i.e.* parallelism, orthogonality and collinearity.



By denoting by $x_i \in [-\theta_{max}, \theta_{max}]$ the quantity to be added to the initial orientation of the line-segment i with respect to its center, we formulate the line-segment re-orientation problem by minimizing the energy

$$U(\mathbf{x}) = (1-\lambda)D(\mathbf{x}) + \lambda V(\mathbf{x}) \qquad (4.1)$$

where $\mathbf{x} = (x_1, .., x_n)$ is a configuration of perturbations operated on the n line-segments, $D(\mathbf{x})$ and $V(\mathbf{x})$ represent a data term and pairwise potential respectively, and $\lambda \in [0, 1]$ is a parameter weighting these two terms, typically 0.8 in our experiments.

Data term $D(\mathbf{x})$ discourages strong angle deviations with respect to their initial orientation. It is expressed by

$$D(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i}{\theta_{max}}\right)^2 \qquad (4.2)$$

Pairwise potential $V(\mathbf{x})$ encourages pairs of spatially close line-segments which are nearly-parallel or nearly-orthogonal to be exactly parallel or orthogonal:

$$V(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n}\sum_{j>i}\mu_{ij}}\sum_{i=1}^{n}\sum_{j>i}\mu_{ij}\frac{|\theta_{ij} - x_i + x_j|}{4\theta_{max}} \qquad (4.3)$$

where θ_{ij} measures how far the relative angle α_{ij} between line-segments i and j is from a straight or right angle. Formally, $\theta_{ij} = \alpha_{ij} \pmod{\pi}$ if $\alpha_{ij} \in [-\frac{\pi}{4}, \frac{\pi}{4}[\cup[\frac{3\pi}{4}, \frac{5\pi}{4}[$ and $\theta_{ij} = \alpha_{ij} - \frac{\pi}{2} \pmod{\pi}$ otherwise.

The dummy variable μ_{ij} returns 1 if line-segments i and j are (i) spatially close and (ii) $|\theta_{ij}| < 2\theta_{max}$, and 0 otherwise. We consider that two line-segments are spatially close if, after building a Delaunay triangulation of points regularly sampled on all the line-segments, at least one Delaunay edge connects their respective sampled points. In practice, sampled points are distant by 10 pixels. Note that time for building a Delaunay triangulation is rather negligible with respect to other operations. Such a neighborhood strongly reduces the number of irrelevant interactions with respect to a standard Euclidean distance by imposing direct visibility between line-segments.

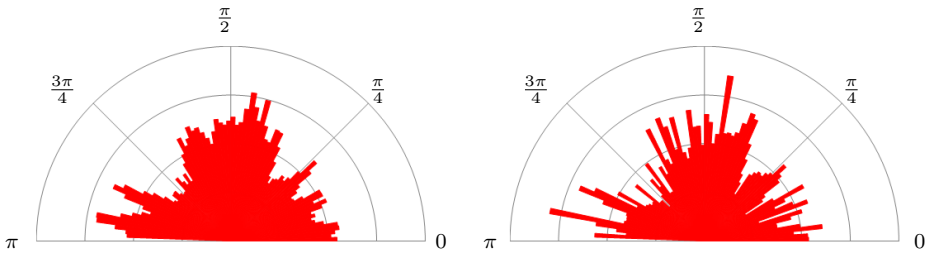


Figure 4.1: Line-segment re-orientation on a 9 Mpixels satellite image of Seoul city. The quite-uniform orientation histogram of initial line-segments (left) makes a few dominant orientations appear after the regularization (right).

Assuming there are m non-zero μ_{ij}, we introduce a new set of variables $\mathbf{y} = (y_1, .., y_m)$ so that our formulation can be turned to a quadratic optimization problem with $(n+m)$ variables and $2(n+m)$ linear constraints:

$$\begin{aligned}
\underset{\mathbf{x},\mathbf{y}}{\text{minimize}} \quad & (1-\lambda)\sum_{i=1}^{n}\left(\frac{x_i}{\theta_{max}}\right)^2 + \lambda\sum_{k=1}^{m} y_k \\
\text{subject to} \quad & x_i \leq \theta_{max}, \quad i = 1, \ldots, n \\
& -x_i \leq \theta_{max}, \quad i = 1, \ldots, n \\
& y_k \leq \frac{1}{4\theta_{max}}(\theta_{ij} - x_i + x_j), \quad k = 1, \ldots, m \\
& -y_k \leq \frac{1}{4\theta_{max}}(\theta_{ij} - x_i + x_j), \quad k = 1, \ldots, m
\end{aligned} \tag{4.4}$$

This minimization problem is solved using a standard optimization library [GW03].

We then use an analogous formulation to re-align line-segments. By now denoting by $x_i \in [-d_{max}, d_{max}]$ the translation to be operated on the line-segment i along its orthogonal vector, we minimize the energy $U(\mathbf{x})$ of Equation 4.4 with $D(\mathbf{x}) = \sum_{i=1}^{n}(x_i/d_{max})^2$ and $V(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j>i}\mu'_{ij}(|d_{ij} - x_i + x_j|/4d_{max})$. Here, d_{ij} corresponds to the distance between the support lines of parallel line-segments i and j, whereas μ'_{ij} returns 1 if (i) $\mu_{ij} = 1$, (ii) line-segments i and j are parallel, and (iii) $d_{ij} < 2d_{max}$, and 0 otherwise. In our experiments, we typically fixed θ_{max} and d_{max} to 5° and 1 pixel. Figures 4.1 and 4.2 show the impact of regularization on urban scenes.

Running times of the regularization procedure vary between a few seconds and a few minutes, depending on the size of the problem. On an indicative

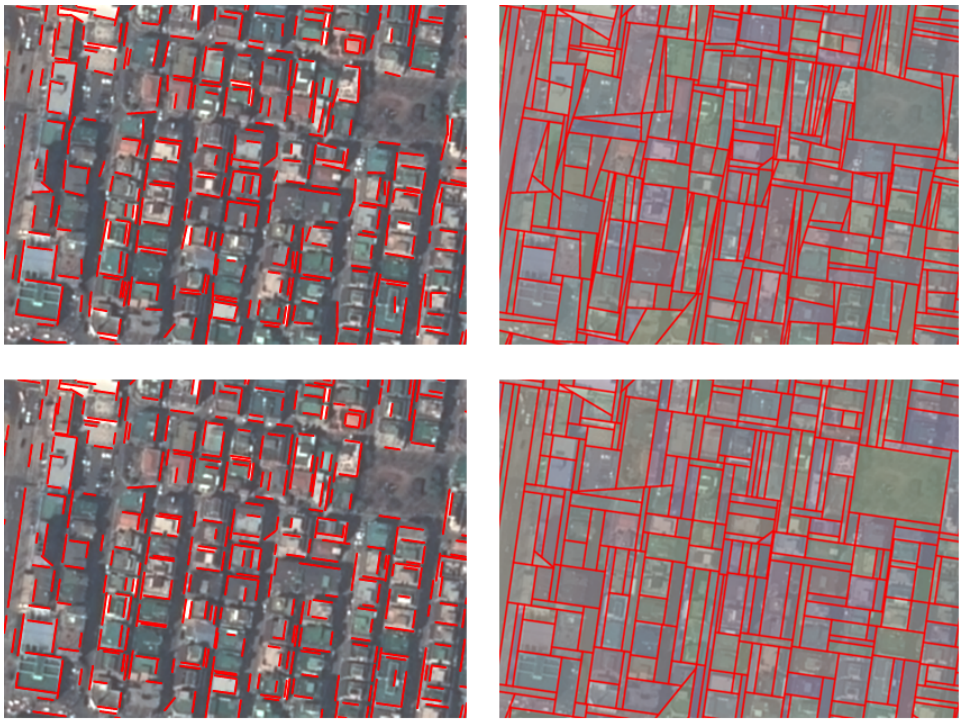


Figure 4.2: Global regularization of line-segments. Floating line-segments detecting by the Line-Segment Detector of Van Gioi *et al* [vGJMR10] (top left) yields a complex polygonal partition with many meaningless polygons (top right). By regularizing them (bottom left), we both simplify the partition with typically around 20% less polygons, and improve the polygon alignments with typical building layouts.

basis, a 9 Mpixels satellite image of Seoul city, whose a cropped part is showed in Figure 4.2, is processed in 13.5 seconds. We refer the reader to the Table 5.1 for more values.

4.2 Plane detection and regularization

4.2.1 Choice of a plane detection technique

To extract planar primitives from 3D data, we use the region-growing implementation of the CGAL library [OVJ+19]. This approach requires a few parameters: a minimal number σ of plane inliers, a fitting tolerence ε to the plane and a normal deviation θ for matching a neighbor vertex to the current processed point.

In our experiments, the fitting tolerance ε and the minimal shape size σ are typically set to 0.2% of the bounding box diagonal and 0.01% of the total number of input points in case of Laser acquisition and synthetic data. The fitting tolerance is increased to 0.5% for multi-view stereo acquisition which generates more noisy point clouds. For approximation of freeform objects, these values vary according to the desired abstraction level: the higher the values of ε and σ, the more concise the output model. θ, for its part, it set to a default value of 25°.

Note that our polyhedral partitioning algorithm does not require planes, but planar polygons. However, obtaining polygons from planes is easy. A possible method only requires to project all inliers onto a plane and compute a convex hull of the projected points.

4.2.2 Regularization of a set of 3D planar primitives

Inspired by our approach in the 2D case, we now introduce a global regularization procedure of the set of planes supporting the planar primitives extracted from an input point cloud. It can correct imprecisions in the planar primitives extraction step, improve the performances of the kinetic partitioning algorithm, and decrease the complexity of the subsequent partition.

Our procedure takes the shape of a two-step process, in which the planes are first re-oriented, then re-aligned, in order to favor geometric relationships between the planes. Like in 2D, both stages of the process are formulated as quadratic optimization problems with linear constraints. The targeted relationships are parallelism, coplanarity, and z-symmetry, and a particular case of orthogonality. Our approach is tailored for point clouds describing man-made structures like buildings, and assumes that the data is correctly oriented in the usual 3D frame.

Let $P = (P_1, P_2, \dots P_N)$ be a set of N detected planes. Each plane P_i is

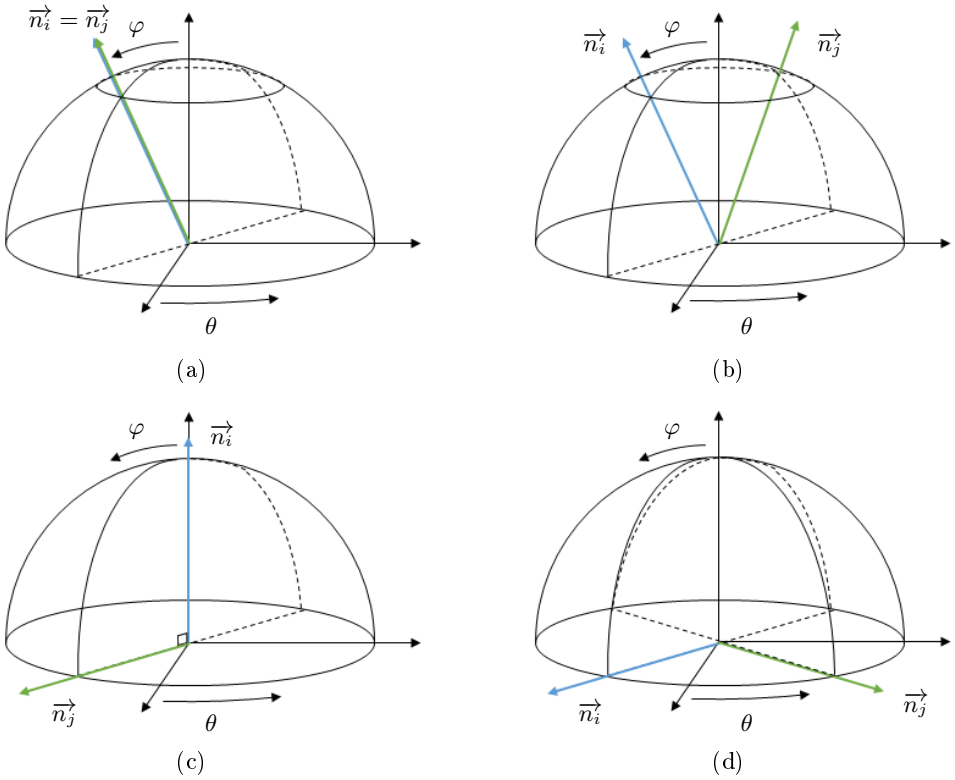


Figure 4.3: Optimized geometric relationships for normal vectors $\vec{n_i}$ and $\vec{n_j}$. (a) Parallelism and coplanarity. (b) Z-symmetry. (c) Orthogonality-1. (d) Orthogonality-2.

represented by a couple $(\overrightarrow{n_i}, d_i)$ where the former is the unit normal vector to the plane, and the latter its signed distance to the origin of the 3D frame.

4.2.2.1 Re-orienting the planes

Each plane P_i is associated to a normal vector $\overrightarrow{n_i}$ that can be converted to a couple of spherical coordinates (φ_i, θ_i) on a half-unit sphere. We obtain the latitude $\varphi_i \in [0, \pi/2]$ and the longitude $\theta_i \in]-\pi, \pi]$. These spherical coordinates can be used to establish one geometric relationship between two adjacent planes P_i and P_j among those listed in Table 4.1.

Note that we only consider a simplified version of orthogonality, since the equation that ties couples (φ_i, θ_i) and (φ_j, θ_j) is not linear in the general case. Given a vector $\overrightarrow{n_i}$, there exists indeed a infinity of vectors $\overrightarrow{n_j}$ that are orthogonal to it. However, this formulation still handles most of the cases

Relationship	Mathematical formulation
Parallelism	$\varphi_i = \varphi_j$ $\theta_i = \theta_j$
Z-symmetry	$\varphi_i = \varphi_j$ $\theta_i - \theta_j \equiv \pi \mod 2\pi$
Orthogonality-1	$\varphi_i - \varphi_j = \pm\pi/2$
Orthogonality-2	$\varphi_i + \varphi_j = 0$ $\theta_i - \theta_j \equiv \pi/2 \mod \pi$

Table 4.1: Optimized geometric relationships and their mathematical pairwise formulations.

observed in the real world, and applies well to buildings in particular.

Energy. Similarly to the 2D case, our reorientation procedure consists in adding quantities $x_{2i-1} \in [-\varphi_{max}, \varphi_{max}]$ and $x_{2i} \in [-\theta_{max}, \theta_{max}]$ to the first and second elements of each couple (φ_i, θ_i) in order to obtain new couples (φ'_i, θ'_i) satisfying one of the equations of Table 4.1. The optimal vector of perturbations $\mathbf{x} = (x_1, \dots x_{2N})$ is then obtained by minimizing the energy

$$U(\mathbf{x}) = (1-\lambda)D(\mathbf{x}) + \lambda V(\mathbf{x}) \tag{4.5}$$

where $D(\mathbf{x})$ is a data term discouraging high perturbations, and $V(\mathbf{x})$ is a pairwise potential encouraging pairs of near-parallel, near-z-symmetric and near-orthogonal planes to become exactly parallel, z-symmetric or orthogonal. As for the balancing term $\lambda \in [0,1]$, it is again typically set to 0.8.

The data term $D(\mathbf{x})$ is defined as:

$$D(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\left[\left(\frac{x_{2i-1}}{\varphi_{max}}\right)^2 + \left(\frac{x_{2i}}{\theta_{max}}\right)^2\right] \tag{4.6}$$

The pairwise potential $V(\mathbf{x})$, for its part, is a normalized sum of four terms, each of them being associated to one of the geometric relationships we would like to optimize:

$$V(\mathbf{x}) = \frac{1}{M}\left(V_P(\mathbf{x}) + V_Z(\mathbf{x}) + V_O^1(\mathbf{x}) + V_O^2(\mathbf{x})\right) \tag{4.7}$$

Indeed, when initializing this optimization problem, we take each pair of spatially close planes P_i and P_j and assign this pair to one of the terms $V_P(\mathbf{x})$, $V_Z(\mathbf{x})$, $V_O^1(\mathbf{x})$ or $V_O^2(\mathbf{x})$, depending on if these planes are near-parallel, near-z-symmetric or near-orthogonal. To this end, we compute variables φ_{ij} and θ_{ij} that indicate how far P_i and P_j are from a given relationship.

Parallelism. Re-oriented planes P_i and P_j become parallel if and only if

$$\iff \begin{cases} \varphi_i + x_{2i-1} &= \varphi_j + x_{2j-1} \\ \theta_i + x_{2i} &\equiv \theta_j + x_{2j} \mod 2\pi \\ x_{2i-1} - x_{2j-1} &= -\varphi_i + \varphi_j \\ x_{2i} - x_{2j} &\equiv -\theta_i + \theta_j \mod 2\pi \end{cases}$$

Let us set:

$$\begin{cases} \varphi_{ij}^P &= -\varphi_i + \varphi_j \\ \theta_{ij}^P &= -\theta_i + \theta_j + 2k\pi, \quad k \in \mathbb{Z} \end{cases}$$

If there exists k such that $|\varphi_{ij}^P| < 2\varphi_{max}$ and $|\theta_{ij}^P| < 2\theta_{max}$, then we mark the planes P_i and P_j as near-parallel and set a boolean variable μ_{ij}^P to 1. Otherwise, it is set to 0. $V_P(\mathbf{x})$, which encourages these planes to become exactly parallel, is defined as:

$$V_P(\mathbf{x}) = \sum_{i=1}^{N} \sum_{j>i} \mu_{ij}^P \left(\frac{|\varphi_{ij}^P - x_{2i-1} + x_{2j-1}|}{4\varphi_{max}} + \frac{|\theta_{ij}^P - x_{2i} + x_{2j}|}{4\theta_{max}} \right) \tag{4.8}$$

Now, assuming that there are μ_P near-parallel planes, we can actually reformulate $V_P(\mathbf{x})$ as a sum of constrained variables:

$$\begin{aligned} V_P(\mathbf{x}) &= \sum_{k=1}^{\mu_P} (y_k + z_k) \\ y_k &\leq \frac{1}{4\varphi_{max}} (\varphi_{ij}^P - x_{2i-1} + x_{2j-1}) \\ -y_k &\leq \frac{1}{4\varphi_{max}} (\varphi_{ij}^P - x_{2i-1} + x_{2j-1}) \\ z_k &\leq \frac{1}{4\theta_{max}} (\theta_{ij}^P - x_{2i} + x_{2j}) \\ -z_k &\leq \frac{1}{4\theta_{max}} (\theta_{ij}^P - x_{2i} + x_{2j}) \end{aligned} \tag{4.9}$$

Z-symmetry. Likewise, planes P_i and P_j become z-symmetric if and only if

$$\iff \begin{cases} \varphi_i + x_{2i-1} &= \varphi_j + x_{2j-1} \\ (\theta_i + x_{2i}) - (\theta_j + x_{2j}) &\equiv \pi \mod 2\pi \end{cases} \iff \begin{cases} x_{2i-1} - x_{2j-1} &= -\varphi_i + \varphi_j \\ x_{2i} - x_{2j} &\equiv -\theta_i + \theta_j + \pi \mod 2\pi \end{cases}$$

We set:

$$\begin{cases} \varphi_{ij}^Z &= -\varphi_i + \varphi_j \\ \theta_{ij}^Z &= -\theta_i + \theta_j + (2k+1)\pi, \quad k \in \mathbb{Z} \end{cases}$$

If there exists k such that $|\varphi_{ij}^Z| < 2\varphi_{max}$ and $|\theta_{ij}^Z| < 2\theta_{max}$, then planes P_i and P_j are considered near-z-symmetric. We set a dummy variable μ_{ij}^Z to 1 and we define $V_Z(\mathbf{x})$ as:

$$V_Z(\mathbf{x}) = \sum_{i=1}^{N} \sum_{j>i} \mu_{ij}^Z \left(\frac{|\varphi_{ij}^Z - x_{2i-1} + x_{2j-1}|}{4\varphi_{max}} + \frac{|\theta_{ij}^Z - x_{2i} + x_{2j}|}{4\theta_{max}} \right) \tag{4.10}$$

Assuming that we identified μ_Z planes as near-z-symmetric, $V_Z(\mathbf{x})$ can also be rewritten as a sum of constrained variables:

$$\begin{aligned} V_Z(\mathbf{x}) &= \sum_{k=1}^{\mu_Z} (y_k + z_k) \\ y_k &\leq \frac{1}{4\varphi_{max}} (\varphi_{ij}^Z - x_{2i-1} + x_{2j-1}) \\ -y_k &\leq \frac{1}{4\varphi_{max}} (\varphi_{ij}^Z - x_{2i-1} + x_{2j-1}) \\ z_k &\leq \frac{1}{4\theta_{max}} (\theta_{ij}^Z - x_{2i} + x_{2j}) \\ -z_k &\leq \frac{1}{4\theta_{max}} (\theta_{ij}^Z - x_{2i} + x_{2j}) \end{aligned} \tag{4.11}$$

Orthogonality-1. In this particular case of orthogonality, only latitudes are taken into account. Planes P_i and P_j become orthogonal if and only if:

$$\begin{aligned} & (\varphi_i + x_{2i-1}) - (\varphi_j + x_{2j-1}) = \pm\tfrac{\pi}{2} \\ \iff \quad & x_{2i-1} - x_{2j-1} = \pm\tfrac{\pi}{2} - \varphi_i + \varphi_j \end{aligned}$$

Similarly to above, we define a variable:

$$\varphi_{ij}^{O_1} = \pm\frac{\pi}{2} - \varphi_i + \varphi_j$$

If $|\varphi_{ij}^{O_1}| < 2\varphi_{max}$, then planes P_i and P_j are considered as near-orthogonal and the boolean variable $\mu_{ij}^{O_1}$ is set to 1. We define:

$$V_O^1(\mathbf{x}) = \sum_{i=1}^{N}\sum_{j>i}\mu_{ij}^{O_1}\frac{|\varphi_{ij}^{O_1} - x_{2i-1} + x_{2j-1}|}{4\varphi_{max}} \tag{4.12}$$

Assuming that there exists μ_{O_1} near-orthogonal pairs of planes considered by this function, we get:

$$\begin{aligned} V_{O_1}(\mathbf{x}) &= \sum_{k=1}^{\mu_{O_1}} y_k \\ y_k &\leq \frac{1}{4\varphi_{max}}(\varphi_{ij}^{O_1} - x_{2i-1} + x_{2j-1}) \\ -y_k &\leq \frac{1}{4\varphi_{max}}(\varphi_{ij}^{O_1} - x_{2i-1} + x_{2j-1}) \end{aligned} \tag{4.13}$$

Orthogonality-2. Finally, the other case of orthogonality applies to the reoriented planes P_i and P_j when:

$$\iff \begin{cases} (\varphi_i + x_{2i-1}) + (\varphi_j + x_{2j-1}) &= 0 \\ (\theta_i + x_{2i}) - (\theta_j + x_{2j}) &\equiv \frac{\pi}{2} \mod 2\pi \end{cases} \iff \begin{cases} x_{2i-1} + x_{2j-1} &= -\varphi_i - \varphi_j \\ x_{2i} - x_{2j} &\equiv \frac{\pi}{2} - \theta_i + \theta_j \mod 2\pi \end{cases}$$

Finally, we set:

$$\begin{cases} \varphi_{ij}^{O_2} &= -\varphi_i - \varphi_j \\ \theta_{ij}^{O_2} &= \frac{\pi}{2} - \theta_i + \theta_j + 2k\pi, \quad k \in \mathbb{Z} \end{cases}$$

Again, if there exists k such that $|\varphi_{ij}^{O_2}| < 2\varphi_{max}$ and $|\theta_{ij}^{O_2}| < 2\theta_{max}$, then planes P_i and P_j are considered near-orthogonal. As a consequence, we trigger the corresponding boolean variable $\mu_{ij}^{O_2}$ in the final term $V_O^2(\mathbf{x})$, which is defined as:

$$V_O^2(\mathbf{x}) = \sum_{i=1}^{N}\sum_{j>i}\mu_{ij}^{O_2}\left(\frac{|\varphi_{ij}^{O_2} - x_{2i-1} - x_{2j-1}|}{4\varphi_{max}} + \frac{|\theta_{ij}^{O_2} - x_{2i} + x_{2j}|}{4\theta_{max}}\right) \tag{4.14}$$

Finally, considering that there exists μ_{O_2} pairs of near-orthogonal planes regularized via $V_O^2(\mathbf{x})$, we reformulate the previous equation as:

$$\begin{aligned} V_{O_2}(\mathbf{x}) &= \sum_{k=1}^{\mu_{O_2}} (y_k + z_k) \\ y_k &\leq \frac{1}{4\varphi_{max}} (\varphi_{ij}^{O_2} - x_{2i-1} - x_{2j-1}) \\ -y_k &\leq \frac{1}{4\varphi_{max}} (\varphi_{ij}^{O_2} - x_{2i-1} - x_{2j-1}) \\ z_k &\leq \frac{1}{4\theta_{max}} (\theta_{ij}^{O_2} - x_{2i} + x_{2j}) \\ -z_k &\leq \frac{1}{4\theta_{max}} (\theta_{ij}^{O_2} - x_{2i} + x_{2j}) \end{aligned} \tag{4.15}$$

Normalization term. The normalizing term of equation 4.7 is finally defined as the number of pairs of planes taken into account in all the terms $V_P(\mathbf{x})$, $V_Z(\mathbf{x})$, $V_O^1(\mathbf{x})$ and $V_O^2(\mathbf{x})$. In other words:

$$M = 2\mu_P + 2\mu_Z + \mu_{O_1} + 2\mu_{O_2} \tag{4.16}$$

Quadratic optimization problem. By regrouping the equations 4.6, 4.7, and the systems of inequations 4.9, 4.11, 4.13 and 4.15, minimizing the energy initially introduced in equation 4.5 turns into a quadratic optimization problem of $2N + M$ variables with $4N + 2M$ linear constraints. It can easily be solved using a standard optimization library.

4.2.2.2 Re-aligning the planes

Once planes are re-oriented, we re-align them in order to make pairs of near-coplanar planes exactly coplanar. This is the same problem as in 2D, when pairs of near-collinear line-segments are made exactly collinear.

Let $x_i \in [-d_{max}, d_{max}]$ be the argument of the translation to be operated on the plane P_i, along its normal vector $\overrightarrow{n_i}$. The optimal configuration of arguments $\mathbf{x} = (x_1, x_2, \dots x_N)$ is obtained through the minimization of an energy

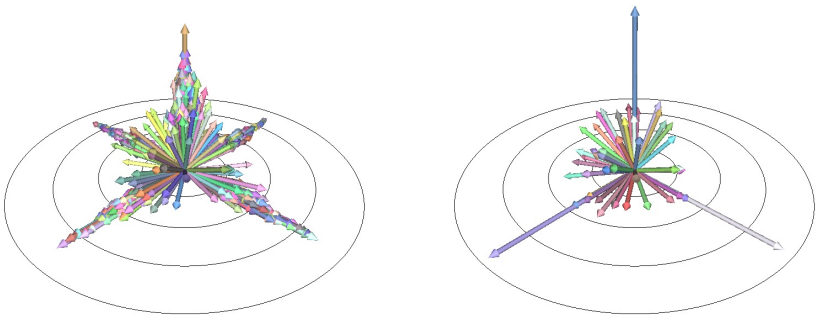


Figure 4.4: Impact of the regularization on the histograms of orientations. Initially, 1096 planes are extracted from the point cloud (left). The regularization procedure reduces this amount to 122 planes (right) in 2.89 seconds, and highlights a few dominant orientations. Model: Hilbert Cube.

$$U(\mathbf{x}) = (1-\lambda)D(\mathbf{x}) + \lambda V(\mathbf{x})$$

$$D(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i}{d_{max}}\right)^2 \tag{4.17}$$

$$V(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{n}\sum_{j>i}\mu_{ij}^{C}\frac{|d_{ij} - x_i + x_j|}{4d_{max}}$$

where

- d_{ij} corresponds to the signed distance between the planes P_i and P_j;
- $\mu_{ij}^{C} = 1$ if and only if (i) $\mu_{ij}^{P} = 1$, (ii) planes P_i and P_j are parallel, and (iii) $|d_{ij}| < 2d_{max}$;
- $M = \sum_{i=1}^{N}\sum_{j>i}\mu_{ij}^{C}$.

In our experiments, we typically set φ_{max} and θ_{max} to 5° and d_{max} to the fitting tolerance ε defined during the plane detection phase.

Figure 4.4 and 4.5 illustrate the effects of the regularization on the reconstruction pipeline. To this end, we apply a random vertex displacement to the vertices of a CAD model from the Thingi10k database [ZJ16], Hilbert Cube and produce two models, with or without applying the regularization.

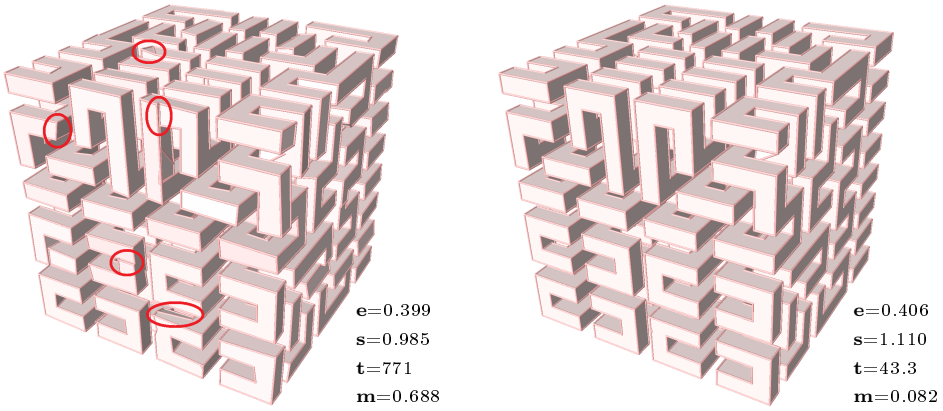


Figure 4.5: Impact of the regularization on the reconstructed models. Left: non-regularized version. Red ellipses highlight asperities and irregularities visible on the surface. in the Right: regularized version. **e**, **s**, **t** and **m** represent respectively the geometric error, the simplicity index, the running times and memory peaks of the subsequent kinetic partitioning algorithm. **t** and **m** are given in seconds and gigabytes. Model: Hilbert Cube.

From a visual point of view, enabling the regularization returns a simplified version of the model by removing asperities noticed in the non-regularized surface. Consequently, the geometric error increases, with respect to the noised CAD model. However, the performances of the polyhedral partitioning algorithm are significantly improved. The regularization procedure itself is quite fast, with an execution time of 2.89 seconds for this example involving 1096 initial planes. Like in 2D, our implementation relies on the OOQP library [GW03].

In our experiments, we apply this regularization procedure when the point cloud describes a man-made object or a urban scene. In average, applying the regularization on man-made objects leads to the generation of partitions that contain 10 to 20% less cells than the non-regularized partitions. However, it requires point clouds to be coherently oriented with respect to the usual 3D frame.

4.3 Conclusions

In this chapter, we introduced a generic, global and efficient regularization procedure for improving a set of basic geometric relationships between a set of line-segments detected in an image. This procedure is formulated as a quadratic optimization problem in the real domain with linear constraints, and can be solved in a short lapse of time. We gave an overview of its impact in the subsequent partitioning step of our vectorization pipeline.

Likewise, we presented a global regularization procedure for improving the geometric consistency of a set of planes, and showed how it can be used to improve the quality of reconstructed models for man-made objects. To this end, we assume that the data is consistently oriented with respect to the usual 3D frame.

In next chapters, we describe a kinetic approach for generating decompositions of 2D or 3D spaces from a set of predetected shapes. We will show that our approach produces lighter and simpler partitions with respect to naive, exhaustive approaches.

Chapter 5

Polygonal partitioning in 2D

In this chapter, we present an algorithm that constructs a 2D polygonal partition from a pre-detected set of line-segments in an image. Polygonal partitions offer a good alternative to traditional superpixels, especially for analyzing scenes that have strong geometric signatures. Current applications of polygonal partitions include for instance image-based rendering [RBDD18], scene illumination [DAPP17], city modeling [DL16] or indoor reconstruction [MMV+14].

Existing algorithms produce homogeneously-sized polygons that fail to capture thin geometric structures and over-partition large uniform areas. We propose a kinetic approach that brings more flexibility on polygon shape and size. The key idea consists in progressively extending pre-detected line-segments until they meet each other. Our experiments demonstrate that output partitions both contain less polygons and better capture geometric structures than those delivered by existing methods.

5.1 Background on kinetic data structures

A kinetic data structure consists in a set of geometric *primitives*, whose coordinates are continuous functions of time. The purpose of kinetic frameworks [BGH99, Gui04] is to maintain the validity of a set of statements that apply to such a data structure. These statements, called *certificates*, are built upon *predicates*, which are functions of the geometric primitives that return a discrete set of values. Most often, predicates evaluate the sign of an algebraic expression binding two primitives or more, and therefore convey an idea of interaction between them.

As primitives move, *events* may occur when certificates become invalid. Kinetic frameworks show a strong algorithmic interest to dynamically order the times of occurrences of the events within a *priority queue*. When an event actually happens on top of the priority queue, the geometric objects

responsible for a certificate failure and the priority queue itself are updated, so that the kinetic data structure remains valid at any time of the simulation.

Examples of kinetic data structures include dynamic Delaunay triangulations of a set of moving vertices [AGG+10], or polyhedral surface reconstruction from point clouds [BBPDM08].

In what follows, we address the problem of image partitioning using the terminology introduced by Basch and Guibas in the aforementioned articles, leading us to design our own kinetic framework, later used to generate the desired partitions.

5.2 Algorithm

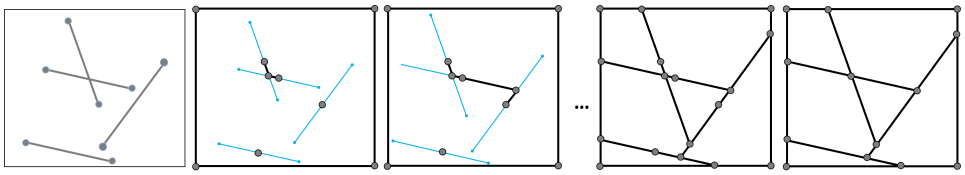


Figure 5.1: Illustration of the kinetic partitioning mechanism. Detected line-segments (a) are converted into an initial planar graph (b). As extremities of primitives extend (blue dots), they meet each other, which enriches the planar graph with new nodes and edges (c, d). After the last collision, the planar graph is simplified by removing all unnecessary nodes (e).

We propose a kinetic framework in which the line-segments are progressively lengthening in the image domain. The underlying data structure is a dynamic planar graph $G_t = (V_t, E_t)$ that partitions the image domain, with V_t and E_t the set of vertices and edges respectively at time t. When line-segments intersect, the complexity of the graph evolves with typically the insertion of new vertices and edges so that it remains planar. We define below the primitives, certificates and update operations of our kinetic formulation.

Primitives. Because the two extremities of a line-segment should be able to expand independently, our primitives correspond to half line-segments. Formally, a detected line-segment between points A and B generates two

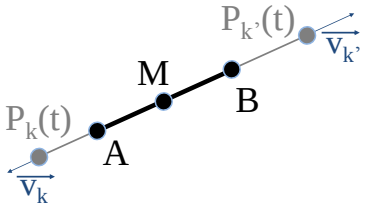


Figure 5.2: Initialisation. A line-segment $[AB]$ is split into two primitives $s_k(t) = [MP_k(t)]$ and $s_{k'}(t) = [MP_{k'}(t)]$, of speed vectors $\overrightarrow{v_k}$ and $\overrightarrow{v_{k'}}$, respectively. These primitives are later integrated to our kinetic data structure.

primitives $s_k(t) = [MP_k(t)]$ and $s_{k'}(t) = [MP_{k'}(t)]$ where fixed point M is the mid-point of A and B, and moving points $P_k(t)$ and $P_{k'}(t)$ evolve with time such that

$$P_k(t) = A + \overrightarrow{v_k} \times t \tag{5.1}$$

$$P_{k'}(t) = B + \overrightarrow{v_{k'}} \times t \tag{5.2}$$

where $\overrightarrow{v_k}$ (respectively $\overrightarrow{v_{k'}}$) is the speed vector of primitive $s_k(t)$ (resp. $s_{k'}(t)$) of direction $\overrightarrow{MA}$ (resp. $\overrightarrow{MB}$) and intensity v_k (resp. $v_{k'}$). In our experiments, v_k is set to 1.

Certificates. For each primitive s_i, we define the certificate function $C_i(t)$ as

$$C_i(t) = \prod_{\substack{j=1 \\ j \neq i}}^{N} Pr_{i,j}(t) \tag{5.3}$$

where N is the number of primitives of the kinetic system, and $Pr_{i,j}(t)$ the predicate function that returns 0 when primitive s_i enters in collision with primitive s_j, *i.e.* when the distance from point to line-segment $d(P_i(t), s_j(t)) = 0$, and 1 otherwise. Primitive s_i is called the *source primitive*, and s_j, the *target primitive*. We also call *collision point*, the point located at the intersection of two primitives.

Initialization. We construct the planar graph at $t = 0$ by inserting as vertices (i) the mid-point of each segment, (ii) the four corner points of

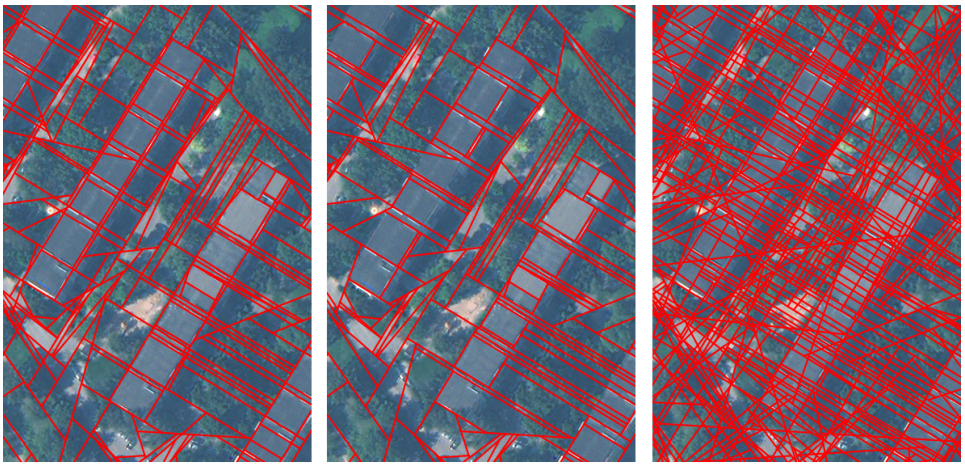
the image domain, and (iii) points located at the intersection of two line-segments, if any. We set edges between the four successive corner points as well as in between possible intersection points and the mid-points of their corresponding line-segments, as illustrated in Figure 5.1-(b).

We also create the priority queue by computing and sorting all the times for which certificates $C_i(t) = 0$ for $i = 1..N$. Instead of considering all possible pairs of line-segments at once, we compute several priority queues in successive time intervals $[kT, (k+1)T[$ to reduce the algorithmic complexity. In practice, when k is incremented, a new priority queue is built from events occurring within this temporal range. By defining the bounding box of a primitive as the smallest image-aligned square that contain the primitive at time $(k+1)T$, and by assuming primitives extend at constant speed, we easily find these events as the pairs of primitives whose bounding boxes overlap. T is fixed to 50 in our experiments, which is a good compromise between running time and memory consumption.

Updating operations. The planarity property of our graph is broken when an event happens, *i.e.* when one of the N certificates become null. We repair it by first inserting the collision point in the graph. When three primitives or more are concurrent, we do not insert this point if it already exists. We then update the edge set of the graph by (i) inserting a new edge between the collision point and the last collision point of the source primitive, and (ii) splitting the edge supporting the target primitive with respect to the collision point, as illustrated in Figure 5.1-(c).

In addition to graph updates, we also decide whether the source primitive should keep propagating. We stop the propagation of the source primitive if it has entered into collision more than a user-defined number of times K, or else if its potential prolongation aligns well with high gradients in the input image. This second condition allows us to not stop the primitive when an obvious image discontinuity along its supporting line exists. Note that our kinetic data-structure is a motorcycle graph [EE99] when $K = 1$ and the gradient-based condition is deactivated. Figure 5.3 shows the impact of the stopping conditions on the output partition.

Finally we update the priority queue by removing the processed event from it, and also, in case the propagation is stopped, all the events created from the certificate function of the source primitive.



(a) (b) (c)

Figure 5.3: Stopping conditions. Setting K to 1 is sufficient to capture the different parts of buildings (a). Deactivating the gradient-based condition leads to omit a few structural components (b) whereas fixing K to a too high value, here 20, gives a too complex partition in which polygons are not meaningful anymore (c).

Finalization. Once the priority queue is empty, we simplify the planar graph by removing the unnecessary vertices, *i.e.* vertices adjacent to two colinear edges which are thus merged, as illustrated in Figure 5.1-(e). Optionally, we also remove skinny polygons when the width of their oriented bounding rectangles is lower than 2 pixels. Such polygons, that can hardly be exploited by subsequent tasks, are merged to the biggest adjacent polygon under the condition the new polygon is convex.

Algorithm 1 Pseudo-code of the Kinetic partitioning

```
Initialize the planar graph G
Initialize the priority queue Q
while Q ≠ ∅ do
    Pop the source and target primitives from Q
    Update G
    Test the stopping condition of the source primitive
    Update Q
end while
Finalize the planar graph
```

The global mechanism of our algorithm is summarized in Algorithm 1. In all our experiments, we set the maximal number of collisions K to 1, except for Figure 5.3. Note that the returned polygons are convex by construction. As concavities inside polygons only appear when two non-colinear primitives intersect at exactly the same time during the propagation phase, we simply force one of the two primitives to keep propagating. Convexity is an interesting property that makes some geometric computations simpler as polygon intersection, point sampling or Constructive Solid Geometry operations. This is for instance useful in 3D reconstruction [BSRVG14]. However, to obtain non-convex polygons, a possible postprocessing could be to group adjacent convex polygons following a color metric.

5.3 Experiments

We tested our algorithm on large-scale satellite images, aerial images, as well as on the Berkeley dataset [MFTM01]. We deactivated the line-segment regularization for Berkeley images which are mainly composed of organic shapes.

5.3.1 Control on the complexity of the partition

Our main parameter is the LSD scale parameter, which allows us to control the sensitivity to image noise, and thus the amount of input line-segments. Despite our algorithm does not offer an exact control on the output number of polygons, this parameter directly impacts on it, as illustrated in Figure 5.4.

5.3.2 Comparison with superpixel methods

We compared our algorithm with state-of-the-art superpixel methods SNIC [AS17] and ERS [LTRC11] and polygonal partitioning methods VORONOI [DL15] and SNICPOLY [AS17]. Because these methods are designed to produce homogeneously-sized regions, our output partitions are visually different, combining both large polygons on homogeneous image areas and thin polygons on lineic structures as illustrated in Figure 5.5. Among the tested methods, only ERS offers enough flexibility on region shapes to capture lineic structures like us. However, converting ERS superpixels into polygons is a delicate task because of boundary irregularities and region connectivity am-

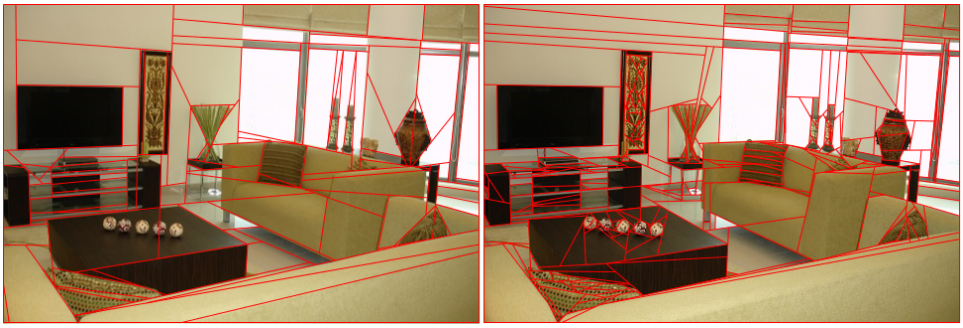


Figure 5.4: Partition complexity. We can produce partitions with varying numbers of polygons by tuning the sensitivity of the line-segment detector. Left partition with 113 polygons is sufficient for capturing the indoor structure and the main furnitures. Right partition (365 polygons) also captures smaller details as some patterns of the background painting.

biguities. Our algorithm performs best on man-made scenes in which objects or object parts can be well captured by polygons.

We evaluated our algorithm on the Berkeley300 dataset [MFTM01] using standard quality criteria of superpixel methods, in particular the boundary recall as defined in [NP12] as well as the boundary precision. The former indicates the ratio of Ground Truth contours correctly recovered by the output region boundaries, whereas the latter measures the ratio of output region boundaries that correctly recovers the Ground Truth contours. We measured the boundary precision on the entire image, contrary to some works [AS17] that compute it on an ε-domain around the Ground Truth contours. For measuring the quality criteria from polygonal partitions, the edges of floating polygons have been discretized into pixel boundaries. We measured these criteria for partitions returning between 50 and $1,000$ regions.

Figure 5.6 shows our algorithm outperforms polygonal partitioning methods VORONOI and SNICPOLY on boundary recall by quite a big margin as their scores at a given number of polygons remain lower than ours with twice less polygons. Our algorithm performs best for a number of regions between 400 and 800, with a boundary recall even higher than superpixel method SNIC. Because our partitions contain large-sized polygons, our algorithm even outperforms superpixel methods on the precision to recall curve when recall is higher than 0.85. To get homogeneously-sized polygons, we can apply a Poisson-disk sampling as postprocessing, similarly to [DL15]. Its

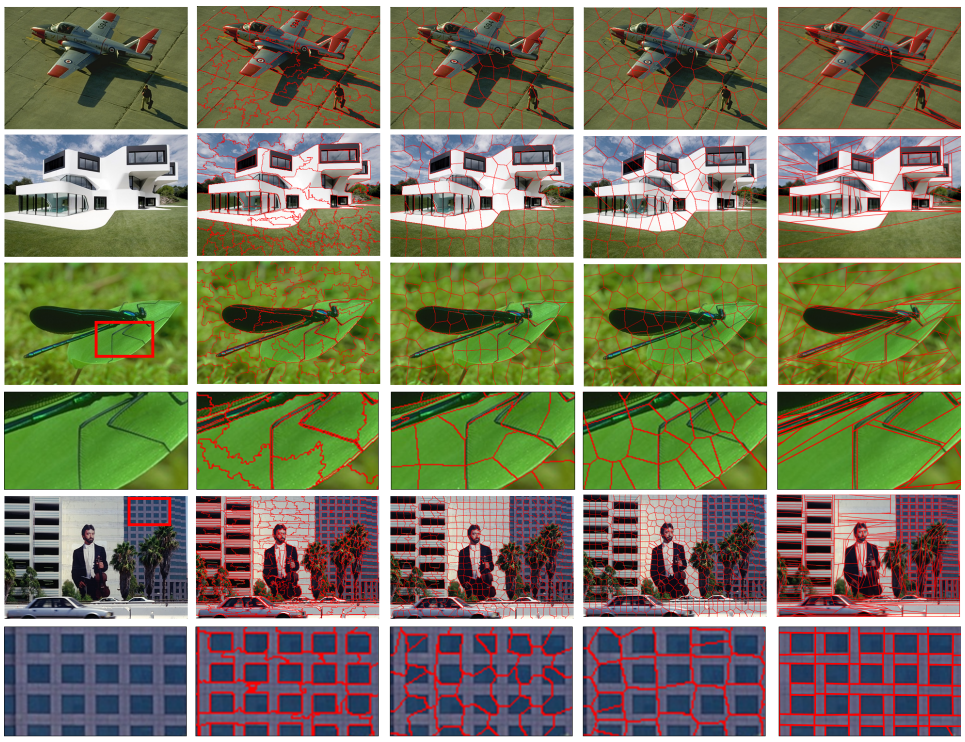


Figure 5.5: Visual comparisons with superpixel and polygonal partitioning methods. Contrary to existing methods designed to deliver homogeneously-sized regions, our partitions combine large polygons capturing homogeneous areas, as the shadow under the airplane, and thin polygons describing lineic structures as the legs of the dragonfly. For an identical number of output regions, our algorithm produces more meaningful polygons, as those capturing the windows of the facade image.

effects on the boundary recall are shown through the curve KIPPI-HOMO: the recall decreases but remains higher than SNICPOLY and VORONOI. When deactivating Poisson disk sampling on VORONOI, the boundary recall improves by a few hundredths but remains lower than SNICPOLY as shown with the curve VORONOI-HETERO.

By reasoning at the scale of geometric shapes instead of pixels, and by exploiting an efficient framework based on Computational Geometry, our algorithm is computationally efficient and scalable. As shown in Table 5.1,

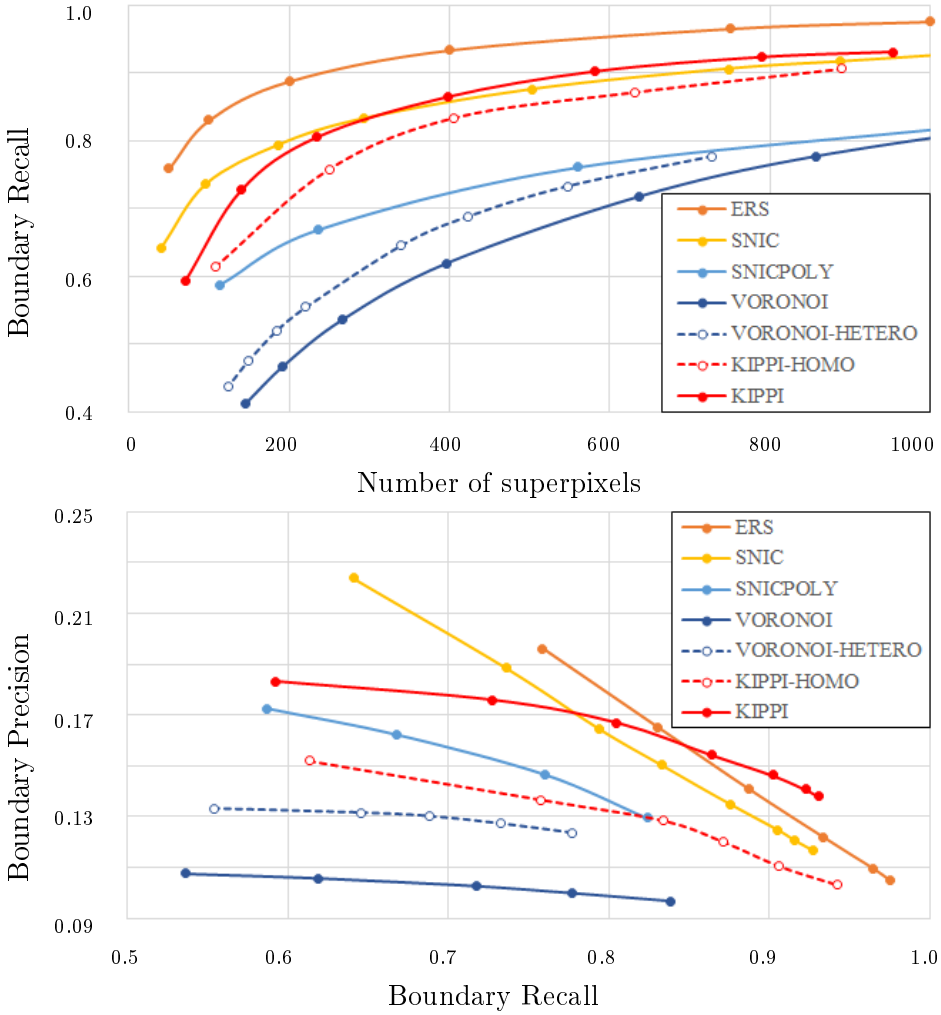


Figure 5.6: Quantitative evaluation. Our algorithm outperforms polygonal partitiong methods VORONOI and SNICPOLY on boundary recall while approaching the scores of the best superpixel methods. Because we allow polygons to be large for capturing big homogeneous areas, our algorithm offers a better compromise between boundary precision and boundary recall than other methods when recall is high, *i.e.* higher than 0.85.

a few minutes are necessary to process a massive satellite image of several hundred millions pixels on a single standard computer. In terms of storage, polygons and their connectivity can be saved in a very compact way with a planar graph.

	Facade 154k pixels	Aerial 2.46M pixels	Satellite 106M pixels
# Line-segments	847	3178	171.1k
# Output polygons	530	2488	124.5k
Line-segment detection	52.4 ms	0.59 s	70.7 s
Regularization	72.8 ms	0.35 s	654.5 s
Kinetic partitioning	51.2 ms	0.23 s	45.1 s
Total time	0.195 s	1.41 s	795.6 s

Table 5.1: Performances on three different image sizes (Facade from Figure 5.5 -bottom, Aerial from Figure 7.2-right, and Satellite whose a cropped part is illustrated on Figure 5.7)-top in terms of running time.

5.3.3 Results on aerial and satellite imagery

We also tested our algorithm on urban environments, using data produced by satellite or aerial imagery. For satellite images, the data was collected by the WorldView-2 and WorldView-3 satellites, therefore the resolution of the images varies between 30 and 50 centimeters per pixel. Aerial images were acquired thanks to UAVs. Assembled orthophotos could be accessed thanks to the OpenAerialMap database [Ope19], and have a standard resolution of 4 centimeters. Figures 5.7 and 5.8 show results generated by our algorithm on these two kinds of images. We observe that most individual structures are captured by subsets of polygonal cells. This assessment opens the door to the extraction of regions of interest, here the building rooftops, as polygonal shapes.

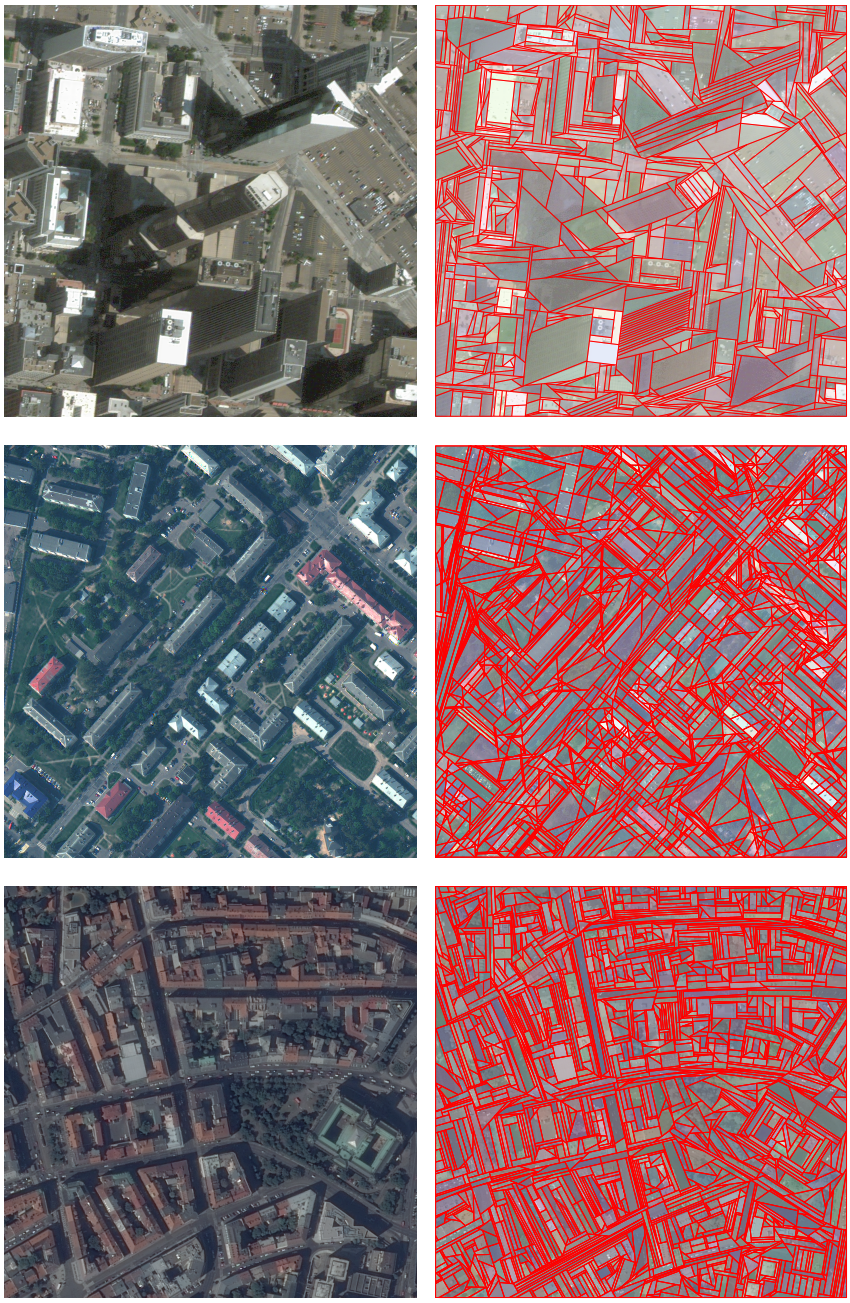


Figure 5.7: Results on satellite images. From top to bottom: Denver (US-CO), Minsk (BY), Prague (CZ). Our algorithm captures the structures of most observed buildings and could serve as a basis for the vectorization of their shapes. Best displayed on screen.

Figure 5.8: Results on aerial images. Top: El Alto (BO). Bottom: Zanzibar (TZ). Images taken from [Ope19]. (Continued next page.)

Figure 5.8: Results on aerial images. Top: New York (US-NY). Bottom: Kokomo (US-IN). Images taken from [Ope19]. We observe that most building rooftops can be approximated as simple subsets of polygonal cells. This paves the way for the vectorization of such objects using a saliency map.

5.4 Limitations

Our algorithm has a few shortcomings.

First of all, it does not offer to the user an exact control on the number of output polygons.

In addition, the regularization of line-segments is not effective on organic images. It reduces the complexity of the partition, but at the expense of accuracy.

Moreover, missing line-segments on small structural parts can lead to under-segmentation situations that are currently not handled by our algorithm. One solution would be to split polygons with heterogeneous radiometry within the kinetic data structure, or as a postprocessing step. Also, all line-segments propagate in the same way, regardless of their relevance.

5.5 Conclusion

We presented a kinetic approach to partition images into floating polygons. Whereas some existing methods impose homogeneously-sized polygons in the style of superpixels, our line-segment extension mechanism offers more flexibility on polygonal shapes. This allows us to better recover geometric matters contained in man-made and organic images, and capture thin structures without over-partitioning large homogeneous areas. By reasoning at the scale of geometric shapes, instead of pixels, within a computational geometry framework, our algorithm is scalable and computationally efficient.

In chapter 7, we show that our partitions can be used to approximate objects of interest, such as buildings in images or more organic shapes, as simple polygons with a low number of edges.

Chapter 6

Polyhedral partitioning in 3D

Converting point clouds into concise polygonal meshes is an automated manner is an enduring problem in computer graphics.

Existing methods commonly operate by assembling a set of planar shapes, previously detected from the input points, which can be done interactively or automatically. These automatic approaches [CLP10, NW17] operate by slicing the 3D space bounding the input point cloud into a partition of convex polyhedral cells with the infinite supporting plane of each primitive. The output mesh is then extracted by labelling the cells as inside our outside the 3D objects. These approaches yield concise meshes, but are not scalable. Complex objects and scenes, which are composed of more than 100 planar shapes, result in huge partitions and too complex energetic formulations when constructing the mesh.

Inspired by our previous work on the polygonal partitioning of images, this chapter describes a shape assembling method which is at least one order of magnitude more efficient than existing methods, both in time and number of processed shapes. Given a set of convex planar polygons as input, we design a kinetic data structure in the 3D space, in which polygons grow until they collide each other. This simple, yet natural idea produces much lighter and meaningful partitions than exhaustive ones and constitute a good support for the subsequent mesh extraction step.

6.1 Algorithm

Our algorithm takes as input a set of convex planar polygons, and returns as output a partition of the bounding 3D space. It consists of a kinetic data structure [Gui04] in which convex polygons extend at constant speed until colliding with each other. When a collision between two or more polygons occurs, we modify the evolution of these polygons by either stopping their growth, changing their direction of propagation or splitting them. This set

of polygons progressively partitions the 3D space into polyhedra, the final partition being obtained when no polygon can grow anymore. We observe that each facet of the underlying connected 3D graph is therefore a part or an extension of the convex polygons.

An introduction of kinetic data structures has been provided in Section 5.1 of this thesis. In what follows, we describe the ingredients of our kinetic framework for the partitioning of a bounding volume. To our knowledge, we are the first to design and implement a kinetic data structure that collides polygons in the 3D space.

Primitives and kinetic data structure. The primitives of our kinetic framework are polygons whose vertices move at constant speeds along given directions. The initial set of polygons is defined as the convex hulls of the planar shapes. Polygons grow by uniform scaling: each vertex moves in the opposite direction to the center of mass of the initial polygon. The underlying kinetic data structure $\mathcal{P}$ is the set of these polygons that, we assume, do not intersect with each other. When two or more polygons intersect, we modify primitives to maintain valid the intersection-free property.

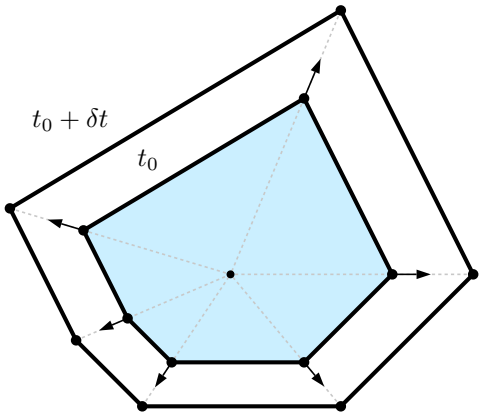


Figure 6.1: A primitive of our kinetic data structure. As a general propagation law, we apply homothetic transformations of ratio $(1+t)$ to the detected planar shapes, where t is the simulation time.

Certificates. For each primitive i, we define the certificate function $C_i(t)$ as

$$C_i(t) = \prod_{\substack{j=1 \\ j \neq i}}^{N} Pr_{i,j}(t) \tag{6.1}$$

where N is the number of primitives of the kinetic data structure, and $Pr_{i,j}(t)$ the predicate function that returns 0 when primitive i collides with primitive j, *i.e.* when the minimal distance between the polygon boundary of primitive i and the closed polygon of primitive j equates zero for the first time, and 1 otherwise. Primitive i is called the *source primitive*, and j, the *target primitive*. Because the directions of propagation of polygons can change along time, the computation of this minimal distance is a difficult task in practice. We detail in Section 6.2 how we compute this distance efficiently.

Initialization. Before starting growing the set of convex hulls, we first need to decompose them into intersection-free polygons. As illustrated in Figure 6.2, each non intersection-free polygon is cut along the intersection lines with the other polygons. Because the propagation of polygons outside the bounding box is not relevant in practice, we also insert the six facets of the bounding box into the kinetic data structure. We denote by $\mathcal{P}_0$ this set of intersection-free polygons. In addition, we populate the priority queue by computing and sorting in ascending order the times of collision, i.e. times for which certificates $C_i(t) = 0$ for each polygon i of $\mathcal{P}_0$.

Updating operations. When a collision between primitives occurs, we need to update the kinetic data structure to keep the set of polygons free of intersection.

We first modify the source polygon. As illustrated in Figure 6.3, four cases are distinguished:

- A vertex of the source polygon collides with the target polygon (case a). We replace the vertex by two sliding vertices that move along the intersection line in opposite directions.

- A sliding vertex of the source polygon collides with the target polygon (case b). We modify the direction of propagation of the sliding vertex to follow the intersection line with the target polygon. In addition, we

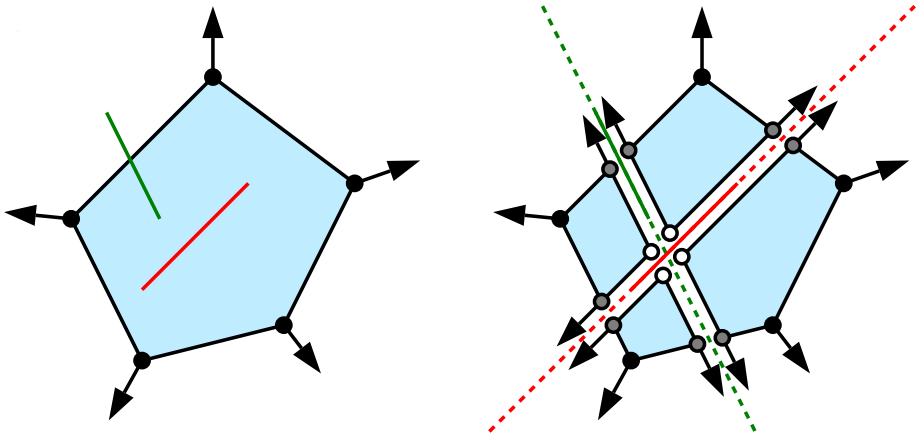


Figure 6.2: Initialization. At $t = 0$, the blue polygon intersects two other polygons, the intersections being represented by the red and green line-segments (left). We decompose it into intersection-free polygons with cuts along the intersection lines (red and green dashed lines). The new vertices are either kept fixed when located at the junction of several intersection lines (white points) or are moving along the intersection lines (gray points). We denote them as *frozen vertices* and *sliding vertices* respectively (right). A polygon is *active* if not all its vertices are frozen.

create a frozen vertex where the intersection line with the target polygon and the intersection line with the polygon supporting the sliding vertex before collision meet.

- A sliding vertex of the source polygon collides with the target polygon while a sliding vertex guided by the target polygon already exists (case c). We create a frozen vertex where the two intersection lines meet: the propagation of the source polygon is locally stopped.

- An edge of the source polygon collides with an edge or vertex of the target polygon (case d). Source and target polygons are split along their intersection line with the creation of eight sliding vertices (two per new polygon).

Note that collisions between two coplanar polygons are included in cases a and b. In some particular situations, it might also happen that an edge of the source polygon collides with the interior of the target polygon. We simply treat it as case a.

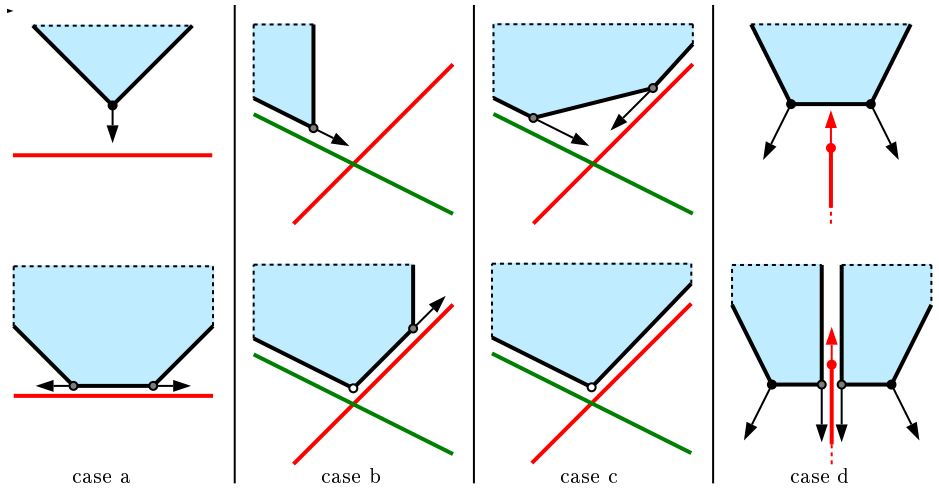
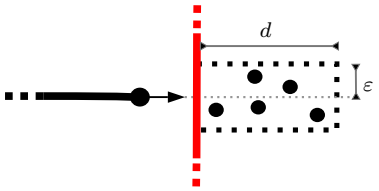


Figure 6.3: Collision typology and corresponding primitive updates. A vertex or an edge of the source polygon (blue shape) typically collides with the target polygon (red segment) in four different manners. The correponding updates operated on the source polygon (bottom) consist in inserting sliding vertices and/or frozen vertices (cases a, b and c) or splitting polygons (case d).

In many situations, it is interesting to extend the source polygon on the other side of the target polygon. To decide whether this should happen, we define a *crossing condition* which is valid if *(i)* the source polygon has collided a number of times lower than a user-specified parameter K, or else if *(ii)* relevant input points omitted during shape detection can be found in the other side of the target polygon. Parameter K is a trade-off between the complexity of the polyhedron partition and the robustness to missing data. In particular, $K = 1$ yields the simplest partition whereas $K = \infty$ produces the partition returned by the exhaustive plane slicing approach. Parameter K is typically set to 2 in our experiments. The second criterion is data-driven: it counts the number of input points contained in a box (black dashed rectangle in the inset) located behind the target polygon (red segment) and aligned with the source polygon (black segment). In the inset, this number is 5. The thickness of the box is twice the fitting tolerance ε used for detecting planar shapes while its depth d is chosen as 10 times the average distance $\widehat{d}$ between neighbors of the



k-nearest neighbor graph of the input points. Assuming the average density of points sampling the object surface is approximately $\widehat{d}^{-2}$, we validate the criterion when the density of points in the box is higher than half of $\widehat{d}^{-2}$ in practice. This data-driven criterion is useful to recover planar parts missed during shape detection, as illustrated in Figure 6.4.

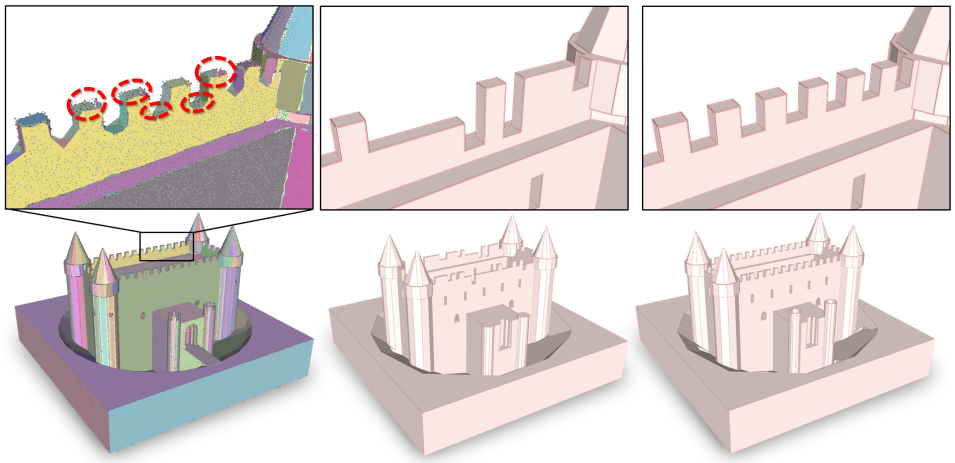
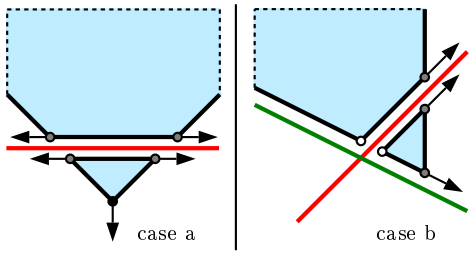


Figure 6.4: Data-driven criterion. Small planar parts are often missed during shape extraction (see missing shapes on the left closeup). The output surface then becomes overly simplified (middle). The data-driven criterion allows us to recover the correct geometry when these planar parts belong to a repetitive structure such as the ramparts (right). Model: CASTLE.

If the crossing condition is valid, a new polygon that extends the source primitive on the other side of the target polygon must be inserted into the kinetic data structure. This situation can only happen to cases a and b in Figure 6.3. The new polygon is initialized as a triangle composed of two sliding vertices and one original vertex for case a, or of two sliding vertices and one frozen vertex for case b (see inset).



Finally we update the priority queue. We first remove the processed event between the source and the target polygons from the queue. If the two polygons have been modified, we recompute their times of collision with the active polygons if not all their vertices are frozen, or remove their times of

collision with the active polygons from the queue otherwise. If new polygons have been created (case d or cases a/b with valid crossing condition), we compute their times of collision with the active polygons and insert them into the priority queue.

Algorithm 2 Kinetic partitioning

```
Initialize the kinetic data structure P to P_0
Initialize the priority queue Q
while Q ≠ ∅ do
    Pop the source and target primitives from Q
    Test the crossing condition of the source primitive
    Update P
    Update Q
end while
Finalize the tessellation
```

Finalization. When the priority queue is empty, the kinetic data structure does not evolve anymore. The output polygons are composed of frozen vertices only and form a partition of polyhedra. We extract this partition as a half-edge data structure by connecting the polygons. Note that the polyhedra are convex by construction. The global mechanism of the kinetic partitioning is summarized in Algorithm 2. A detailed pseudo-code is also provided in appendix.

6.2 Implementation details

Our algorithm has been implemented in C++ using the CGAL library for the geometric operations. In particular, we used the exact predicates, exact constructions kernel for the computation of distances and times of collision. We now explain three important implementation details that improve the efficiency and scalability of our method.

Reformulation as collaborative 2D collision problems. At each update of the kinetic data structure, the propagation of the source polygon is likely to be modified. Consequently, the update of the priority queue requires the distance of this polygon to each other active polygons to be recomputed.

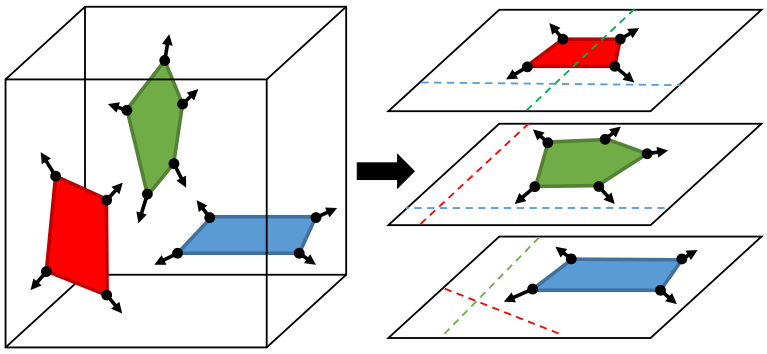


Figure 6.5: Reformulation as collaborative 2D collision problems. The coordinates of N planar polygons in 3D (left) are mapped onto each of their N supporting planes (right). The dashed lines also represent mapped intersection lines between the current supporting plane and another plane. Therefore, events of the kinetic data structure can be computed using a simple point-to-line 2D distance.

To avoid such time consuming operations, we algorithmically reformulate the 3D collision problem as N collaborative propagations of polygons in 2D controlled by a global priority queue, N being the number of planar shapes. The intuition behind this reformulation is that two non-coplanar polygons cannot collide somewhere else than along the 3D line intersecting their respective planes. This 3D line being represented by a 2D line in the planes containing the two polygons, we can then use a simple point-to-line distance in 2D to compute the times of collisions. Although more events must be processed with this reformulation, the update of the priority queue is drastically simplified. Figure 6.5 illustrates our approach.

Priority queue Although all possible collisions should be inserted into the priority queue, we observe that an active vertex often collides with only the three closest lines. To reduce memory consumption and processing time, we thus restrict the number of collisions per vertex in the priority queue to three. If the three collisions occur and the vertex is not frozen we insert the three next collisions of this vertex in the priority queue and repeat this operation as many times as necessary. This reduces running time and memory consumption by a factor close to 1.5 and 3 respectively.

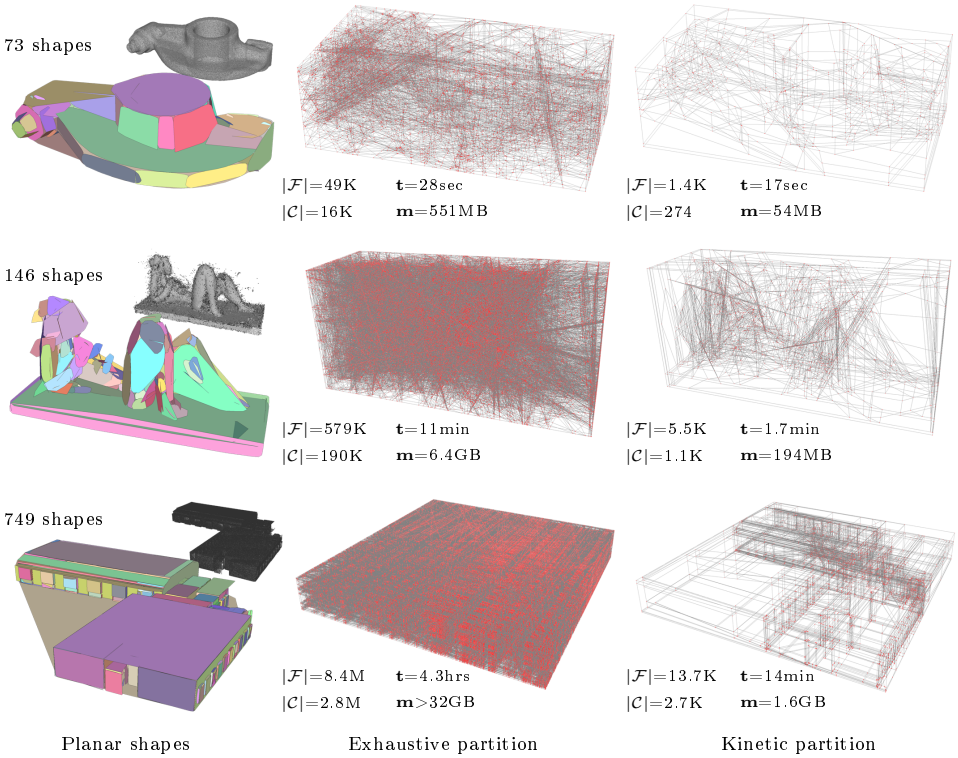


Figure 6.6: Exhaustive vs. kinetic partitionings. By naively slicing all planes supporting the shapes, exhaustive partitions are much more complex than our kinetic partitions. The construction of kinetic partitions is also faster and consumes less memory, especially from complex configurations of planar shapes (see middle and bottom examples). $|\mathcal{F}|$, $|\mathcal{C}|$, $\mathbf{t}$ and $\mathbf{m}$ refer to the number of facets in the partition, the number of polyhedra, the construction time of the partition and the memory consumption respectively. Models from top to bottom: ROCKER ARM, CAPRON and NAVHIS.

Spatial subdivision To increase the scalability of our algorithm, we offer the option to subdivide the bounding box of the object into uniformly-sized 3D blocks in which independent kinetic partitionings are operated. For each block, we collect the closest planar shapes, i.e. those whose initial convex polygon is either inside the block or intersecting at least one of its sides. We then operate the kinetic partitioning inside the block from this subset of planar shapes.

Once all blocks are processed, we merge the polyhedral partitions. An important speed-up factor lies into the fact that only a portion of planar shapes is involved in the partitioning of each block. Our implementation

treats blocks sequentially, but one could process them in parallel for even better performances.

6.3 Experiments

6.3.1 Kinetic vs. exhaustive partitions

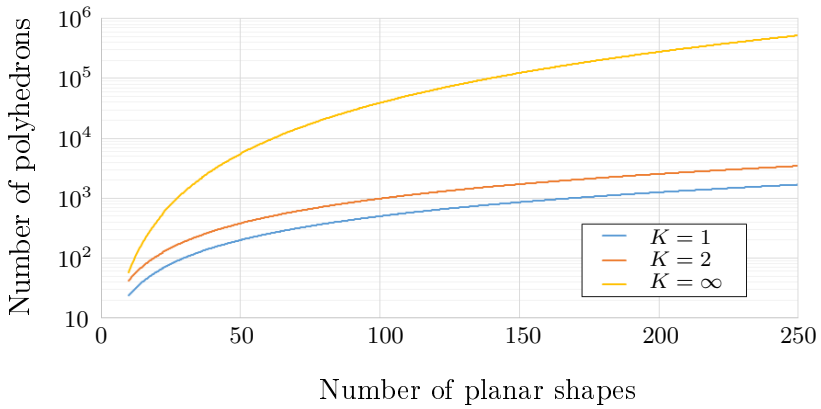
We compare our algorithm of polyhedral partitioning of a 3D space with the naive, exhaustive approach that is implemented in current shape assembling techniques [CLP10, NW17]. Figure 6.7 plots the average number of obtained polyhedra and the running times of both methods for a variety of models, against the number of detected planar shapes.

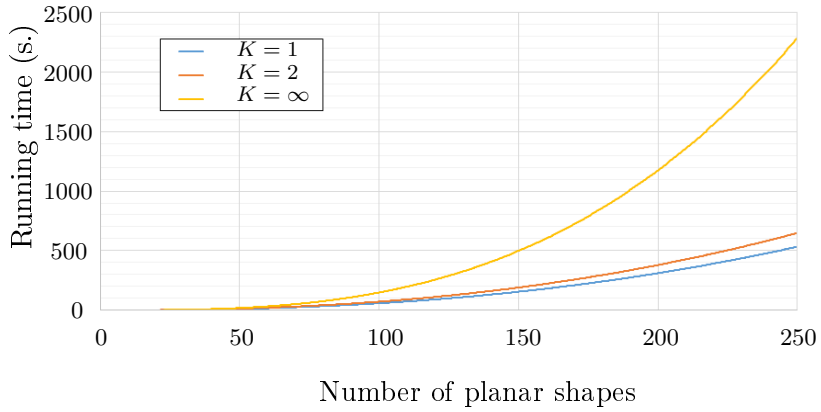
These plots demonstrate the scalability of our kinetic algorithm. For $N = 100$ planar shapes, which approximatively corresponds to the processing limits of the aforementioned techniques, it can return, in average, 100 times less polyhedra and run twice as fast as the exhaustive approach. These factors keep increasing with the number of input planar shapes.

Figure 6.6, which shows examples of kinetic and exhaustive partitions, confirms this trend. Generated in a shorter lapse of time, a kinetic partition is much lighter than an exhaustive one, and contains more meaningful polyhedra.

6.3.2 Spatial subdivision

Figure 6.8 plots the time and memory savings against the number of planar shapes for several subdivision schemes. While the savings are significant, the use of spatial subdivision produces a simplified polyhedral partition as polygons do not propagate to neighboring blocks. Figure 7.10, in next chapter, illustrates this compromise between algorithm efficiency and surface quality.





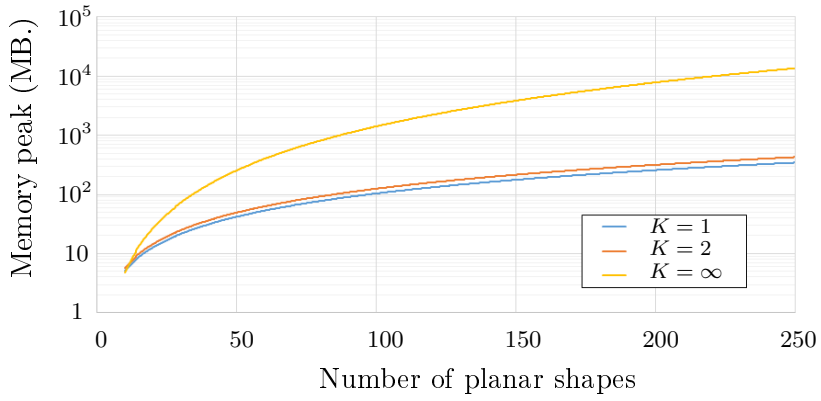


Figure 6.7: Comparative performances between kinetic and exhaustive partitioning in terms of construction time, number of polyhedra, and memory consumption. K represents the maximal number of intersections.

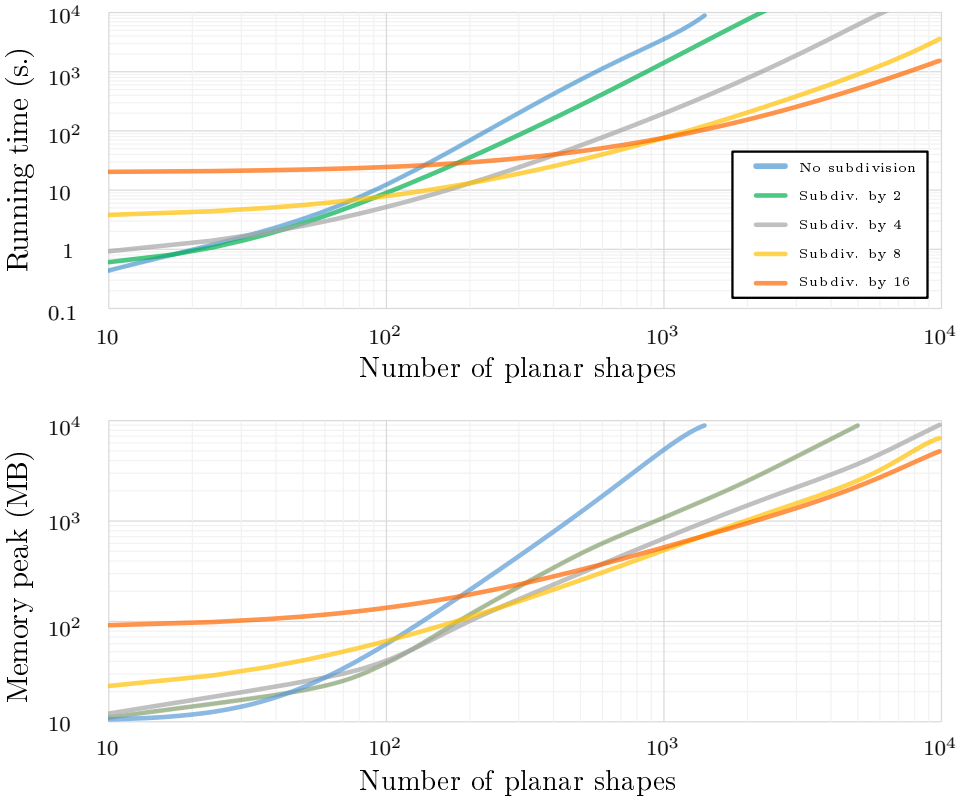


Figure 6.8: Performances with spatial subdivision. The lowest curve in each graph indicates the most efficient subdivision scheme given the number of planar shapes. For instance, one thousand planar shapes can be processed in one minute by subdividing the 3D bounding box into 8^3 blocks (yellow curve in top graph), whereas half an hour is required without subdivision (blue curve). Memory peak evolves similarly to running time. The performances have been obtained from 100k input points uniformly sampled on a sphere.

6.4 Conclusion

We presented an algorithm generates a polyhedral partition of the 3D space using a set of predefined convex planar polygons. Our approach is based on a kinetic data structure, in which these polygons progressively extend until fulfilling a user-defined stopping condition.

By contrast with exhaustive partitions computed in the state-of-the-art shape assembling techniques, our approach generates much lighter partitions.

For 100 detected shapes, which approximatively correspond to the maximal number of shapes that can be digested by these techniques in a reasonable time, a kinetic partition with $K = 1$ contains in average 100 times less polyhedrons than an exhaustive one, and this gap keeps increasing with the number of input shapes. The scalability of our algorithm is further reinforced by a spatial subdivision procedure, that decomposes a complex problem into a set of simpler ones.

In the next chapter, we describe a formulation for producing compact meshes using these partitions, and compare them to the state-of-art polygonal surface reconstruction techniques.

Chapter 7

Object extraction

Chapters 5 and 6 detailed two kinetic algorithms for the polygonal partitioning of a image from a pre-detected set of line-segments, and the polygonal partitioning of a 3D volume based on an initial configuration of planar primitives.

In this chapter, we show how these kinetic partitions can be used for extracting objects and regions of interest in images and point clouds, using a binary cell activation mechanism. The final result is a closed set of edges or facets that approximates these objects by separating the active cells from the inactive ones.

7.1 Object contouring

7.1.1 Description of the model

Object contouring by polygonal shapes provides a compact and structure-aware representation of object silhouettes, in particular in man-made environments [CKUF17, SCF14].

To achieve polygonal object contouring from our partition, we associate each polygon with a binary activation variable indicating if it belongs to the objects of interest or not, similarly to [LSD10] with superpixels. The output polygonal contours correspond to the set of edges separating active polygons from inactive ones, which ensures that the contours are closed by construction.

The problem is formulated as a standard energy minimization problem [BK04]. For each input image, we first compute the probability map H from a few user-provided scribbles, which roughly characterize the radiometric distribution of the foreground objects of interest and the image background. We express the probability $H(i|l)$ of a pixel i to belong to class $l = \{0, 1\}$

as its normalized RGB distance to the closest color in the set of scribbled pixels belonging to that class

$$H(i|l) = \frac{\min_{j \in S_l} \|I(i) - \widehat{I}(j)\|_2^2}{\min_{j \in S_0} \|I(i) - \widehat{I}(j)\|_2^2 + \min_{j \in S_1} \|I(i) - \widehat{I}(j)\|_2^2} \tag{7.1}$$

where S_0 (respectively S_1) is the set of pixels scribbled as foreground (resp. background), and $\widehat{I}$ is the input image convolved by a 11×11 mean filter to remove noise.

Now, let us denote by $\mathcal{F}$ the set of polygonal facets of the graph generated by the partitioning algorithm, and by $x_i \in \{0, 1\}$ a binary activation variable that indicates whether the cell $f_i \in \mathcal{F}$ is considered as a part of an object of interest in the image. We call $\mathbf{x} = (x_i)_{i \in \mathcal{F}}$ the configuration of all binary variables. We minimize the energy $U(\mathbf{x})$ defined as:

$$U(\mathbf{x}) = (1 - \lambda) D(\mathbf{x}) + \lambda V(\mathbf{x}) \tag{7.2}$$

where $D(\mathbf{x})$ is data term measuring the agreement between the binary variable of each polygon and the underlying probability map H, $V(\mathbf{x})$ is a smoothness term based on a Potts model to favor compact contours, and $\lambda \in [0, 1]$ is a parameter balancing these two terms. More precisely, we have:

$$D(\mathbf{x}) = \sum_{i \in \mathcal{F}} \frac{1}{|f_i|} \sum_{p \in f_i} H(p|x_i) \tag{7.3}$$

where $|f_i|$ is the area of the facet f_i and $p \in f_i$ refers to the set of pixels that are included in f_i. The smoothness term is defined, for its part, as:

$$V(\mathbf{x}) = \sum_{i \sim j} l_{ij} \cdot 1_{\{x_i \neq x_j\}} \tag{7.4}$$

where $i \sim j$ denotes pairs of adjacent facets, and l_{ij} is the length of the edge separating facets f_i and f_j.

Note that more advanced methods could be used to predict foreground and background pixels. This would certainly lead to better results, but this is has not been explored within the scope of this thesis.

7.1.2 Experiments and comparisons

Despite the simplicity of our color model H, Figure 7.2 shows our method achieves good results with both organic and man-made shapes. Output poly-

#edges = 65 (28) #edges = 169 (85) #edges = 238 (111)

Figure 7.1: Trade-off between fidelity and simplicity. Polygonal partitions with low complexity give compact polygonal contours that roughly approximate the object silhouette (left). More refined partitions allow us to better capture shape details (right). The numbers between parentheses indicate the final number of edges if we merge all successive collinear edges as a postprocess.

gons capture well the object silhouettes while having a low complexity. In particular, it outperforms results returned by Grabcut [RKB04] followed by a Douglas-Peucker vectorization of the border pixels [DP73]. Replacing our partitions by VORONOI [DL15] reduces accuracy. In particular, VORONOI partitions cannot handle thin structures and tend to produce complex polygonal contours zigzagging around the true silhouettes.

7.1.3 Limitations

Our algorithm of polygonal object contouring seems to return promising results. Thanks to our kinetic partitions, thin structures, such as the legs of the dragonfly in Figure 7.2 are approximated in a satisfactory way, and man-made structures like rooftops can be approximated with a low number of edges.

However, as a cascade pipeline, the results of our algorithm directly depend on the quality of the partitions generated during the previous stage. In particular, cells with heterogeneous contents will decrease the accuracy of our results. A possible strategy would consist in splitting and merging cells based on radiometric or semantic information.

Figure 7.2: Object contouring. Using our partition as input, we are able to capture details in the image missed by other algorithms while producing polygons with lower complexity. Note in particular how thin structures as the legs of the dragonfly or the propeller of the airplane are recovered. Our method performs best on man-made objects composed of piecewise-linear contours, as roofs (right).

In addition, this object contouring model only favors compact contours. It neither explicitely reinforce geometric relations between the selected edges, nor penalizes the lack of regularity of the returned contours. As a result, this object contouring model tends to return too complex and inaccurate results on large partitions with multiple objects of interest, like rooftops in satellite images. This problem can be solved by adding a regularization term to the minimized energy, thus resulting in a more complex formulation.

7.2 Surface extraction

7.2.1 Description of the model

Given a partition of polyhedra, generated by the algorithm described in chapter 6, we wish to extract a surface from it.

We operate a min-cut to find an inside-outside labeling of the polyhedra, the output surface being defined as the interface facets between inside and outside. This strategy garantees the output surface to be watertight

and intersection-free [LPK09]. The methods relying upon this approach [CLP10, BDLGM14, VLA15] typically assign an inside-outside guess on each polyhedron by ray shooting: this is both imprecise in presence of missing data and computationally costly. Instead, we propose a faster voting scheme that exploits the oriented normals of inlier points to more robustly assign a guess on a portion of the polyhedra only.

We denote by $\mathcal{C}$, the set of polyhedra of the tessellation, and by $x_i = \{in, out\}$, the binary label that specifies whether polyhedron i is inside ($x_i = in$) or outside ($x_i = out$) the surface. We measure the quality of a possible output surface $\mathbf{x} = (x_i)_{i \in \mathcal{C}}$ with a two-term energy of the form

$$U(\mathbf{x}) = D(\mathbf{x}) + \lambda V(\mathbf{x}) \tag{7.5}$$

where $D(\mathbf{x})$ and $V(\mathbf{x})$ are terms living in $[0, 1]$ that measure data fidelity and surface complexity respectively. $\lambda \in [0, 1]$ is a parameter balancing these two terms. The optimal output surface that minimizes U is found by a max-flow algorithm [BK04].

Data fidelity $D(\mathbf{x})$ measures the coherence between the inside-outside label of each polyhedron and the orientation of normals of inlier points. Similarly to signed distances proposed in smooth surface reconstruction, we assume here that the normals point towards the outside. After associating each inlier point with the facet of the partition that contains its orthogonal projection on the infinite plane of its planar shape, we express data fidelity by a voting function on each inlier point. More precisely:

$$D(\mathbf{x}) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{C}} \sum_{p \in \mathcal{I}_i} d_i(p, x_i) \tag{7.6}$$

where $|\mathcal{I}|$ is twice the total number of inlier points, $\mathcal{I}_i$ is the set of inlier points associated with all the facets of polyhedron i, and $d_i(p, x_i)$ is a voting function that tests whether the orientation of inlier point p is coherent with the label x_i of polyhedron i.

This function is defined by $d_i(p, in) = 1_{\{\vec{n} \cdot \vec{u} > 0\}}$ and $d_i(p, out) = 1_{\{\vec{n} \cdot \vec{u} < 0\}}$ where $1_{\{.\}}$ is the characteristic function, $\vec{n}$ the normal vector of inlier point p, and $\vec{u}$ the vector from p to the center of mass of polyhedron i. In Figure 7.3, we prefer assigning label in to polyhedron i and label out to polyhedron j. In particular, $d_i(p, x_i = out)$ and $d_j(p, x_j = in)$ return a penalty of 1 whereas $d_i(p, x_i = in) = d_j(p, x_j = out) = 0$. Because the voting function d_i

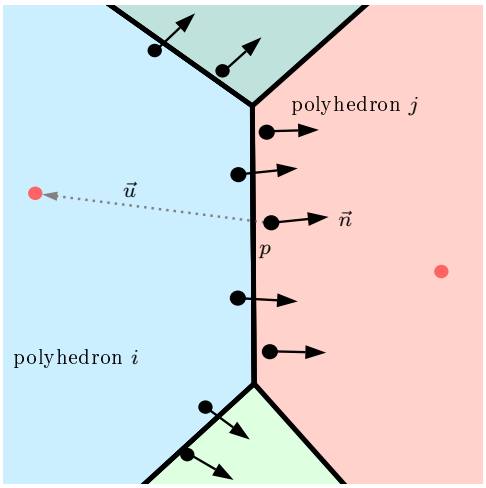


Figure 7.3: Voting criterion.

is binary, normals only need to point towards the right half-space separating the facet. This brings robustness to imprecise normal directions.

The second term $V(\mathbf{x})$ is more conventional: it measures the complexity of the output surface by its area, where lower is simpler. It is expressed by

$$V(\mathbf{x}) = \frac{1}{A} \sum_{i \sim j} a_{ij} \cdot 1_{\{x_i \neq x_j\}} \tag{7.7}$$

where $i \sim j$ denotes the pairs of adjacent polyhedra, a_{ij} represents the area of the common facet between polyhedra i and j, and A is a normalization factor defined as the sum of the areas of all facets of the partition. As illustrated in Figure 7.4, this term avoids the surface zigzagging. Giving a too high importance to this term however shrinks the surface. In our experiments, we typically set parameter λ to 0.5.

7.2.2 Experiments

We evaluated our algorithm with respect to our fidelity, simplicity and efficiency objectives. We measure the fidelity to 3D data by the mean symmetric Hausdorff (MSH) distance between input point cloud and output mesh. The simplicity of output representation is quantified by the ratio between the number of output facets to the number of initial planar shapes. We measure the efficiency of the algorithm by both running time and memory peak.

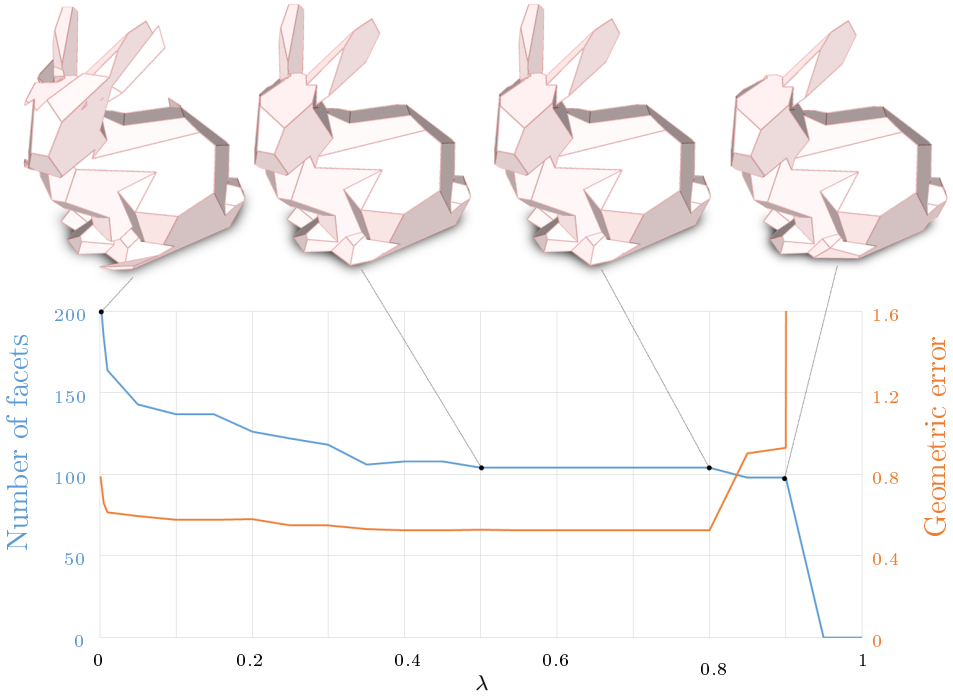


Figure 7.4: Impact of parameter λ. Increasing λ simplifies the output surface by reducing the number of facets (see blue curve). A too high value however makes parts of the object disappear and increases the geometric error (see red curve). The best compromise between simplicity and accuracy is returned in the interval $[0.4, 0.8]$. The geometric error is computed as the symmetric mean Hausdorff distance between input points and output mesh. Model: Stanford bunny.

In addition to these four measures, we also evaluate the scalability of algorithms by the maximal number of planar shapes that can be processed without exceeding both 10^5 seconds on a single computer with a core i9 processor clocked at 2.9Ghz, and 32GB memory consumption.

7.2.2.1 Flexibility

Our algorithm has been tested on a variety of objects, scenes, and acquisition systems.

Our method produces concise polygonal meshes for both freeform objects such as Horse, Ignatius and Asian dragon. Piecewise-planar structures such as Church, Barn and Euler or Full thing are reconstructed with

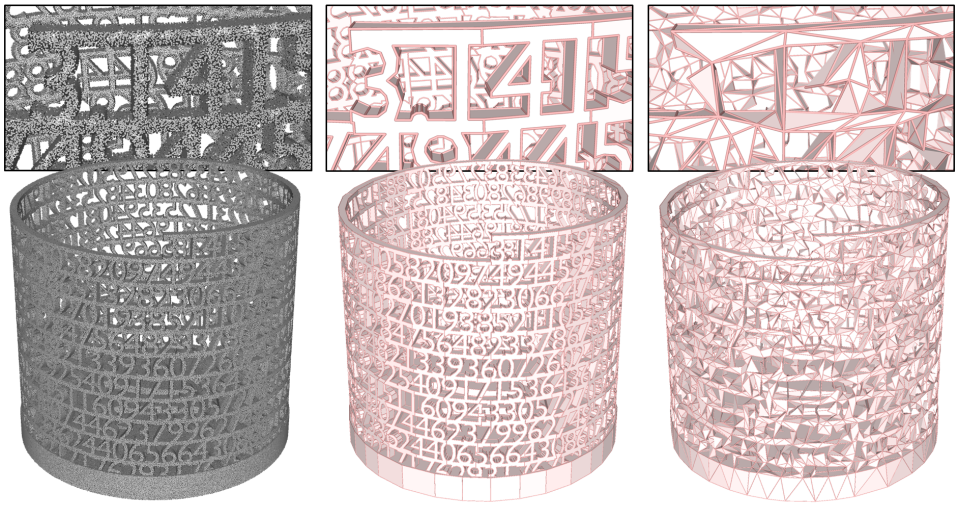
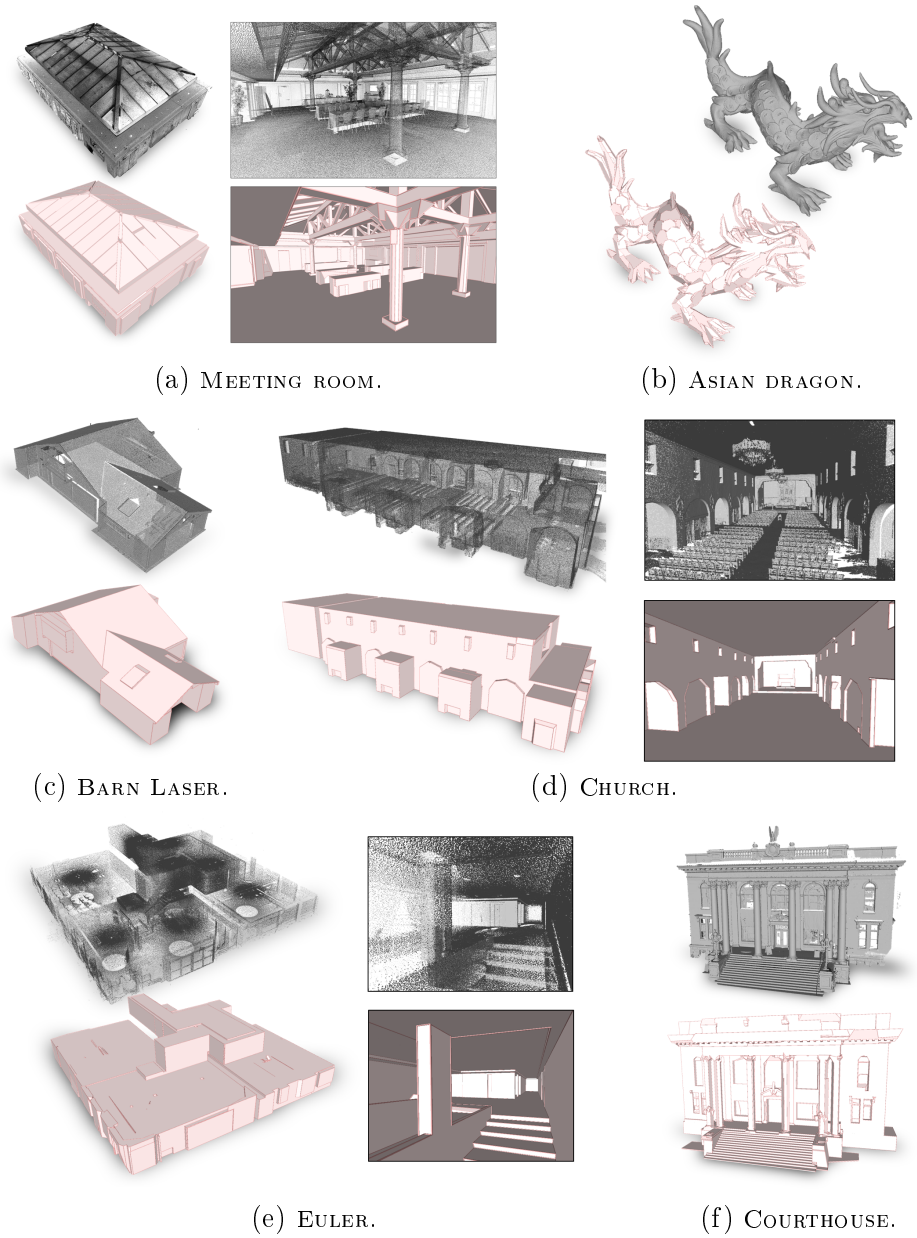


Figure 7.5: Reconstruction of Tower of Pi. $10.8K$ planar shapes are detected from $2.9M$ input points (left) and assembled by our algorithm into a concise polygonal mesh of $12.1K$ facets (middle). Each digit is nicely represented by a few facets (see closeups). In contrast, the mesh produced by the traditional surface approximation pipeline ([KBH06] + [GH97]) fails to preserve the structure of the mesh.

fine details, as long as planar shapes are correctly detected from data. Complex objects such as M60 are approximated with a good amount of details by a few hundred facets mesh only.

Figures 7.6 and 7.7 present reconstructions obtained from laser and Multi-View Stereo (MVS) datasets. The latter have been generated by COLMAP [SF16] from image sequences mostly provided by the Tanks and Temples benchmark [KPZK17]. Because of the high amount of noise, these point clouds are particularly challenging to reconstruct. Point clouds generated by Laser scanning are geometrically more accurate, but suffer from missing data and heterogeneous point density. As illustrated with Barn model, our algorithm typically returns more accurate output meshes with Laser acquisition. MVS point clouds are usually too noisy to capture fine details with small planar shapes. Conversely, frequent occlusions contained in Laser scans are more effectively handled by our kinetic approach that naturally fills in empty space in between planar shapes.

Some point clouds have also been sampled from CAD models. This is the case of Full thing, Castle, Tower of Pi and Hilbert cube whose



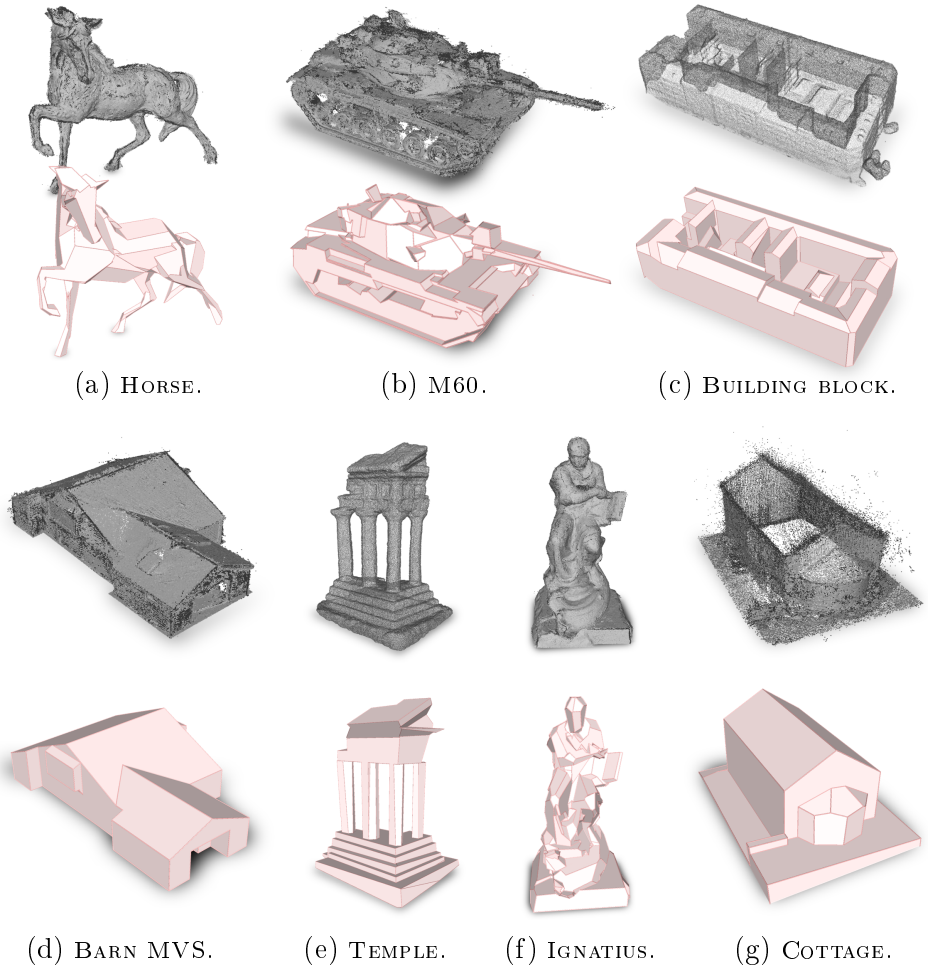
(a) Meeting room. (b) Asian dragon.

(c) Barn Laser. (d) Church.

(e) Euler. (f) Courthouse.

Figure 7.6: Reconstructions from laser datasets. The number of points, detected planar shapes, and facets of the output models are given in Table 7.1.

CAD models originate from the Thingi10k database [ZJ16].



(a) Horse. (b) M60. (c) Building block.

(d) Barn MVS. (e) Temple. (f) Ignatius. (g) Cottage.

Figure 7.7: Reconstructions from multi-view stereo datasets. The number of points, detected planar shapes, and facets of the output models are given in Table 7.1.

	# Input points	# Planar shapes	# Output facets
Meeting room	3.07M	1,655	1,491
Asian dragon	3.61M	2,712	3,132
Barn Laser	613K	151	196
Church	31.1M	1,188	394
Euler	2.73M	887	1,317
Courthouse	1.9M	2,716	1,795
Horse	788K	274	347
M60	2.8M	362	471
Building block	793K	160	142
Barn MVS	619K	95	39
Temple	621K	69	95
Ignatius	1.38M	294	443
Cottage	143K	23	28

Table 7.1: Number of points, planar shapes and output facets for the models presented in Figures 7.6 and 7.7.

7.2.2.2 Robustness to imperfect data

Figure 7.8 shows the robustness of our algorithm to noise. Below 1% noise (with respect to the bounding box diagonal), our algorithm outputs accurate meshes. Above 1%, planar shapes become missing or inaccurately detected. Because our goal is not to correct or complete the shape configuration, the output mesh cannot preserve the structure of the object anymore. Beside noise, the propagation of planar shapes within the kinetic algorithm offers high resilience to occlusions, especially when parameter K is strictly greater than 1. This allows, for instance, to recover the skylights on Barn laser and the rampart structures on Castle.

As illustrated by Figure 7.9, our algorithm is also robust to heterogeneous sampling and outliers by inheriting the good behavior of shape detection methods with respect to these two defects.

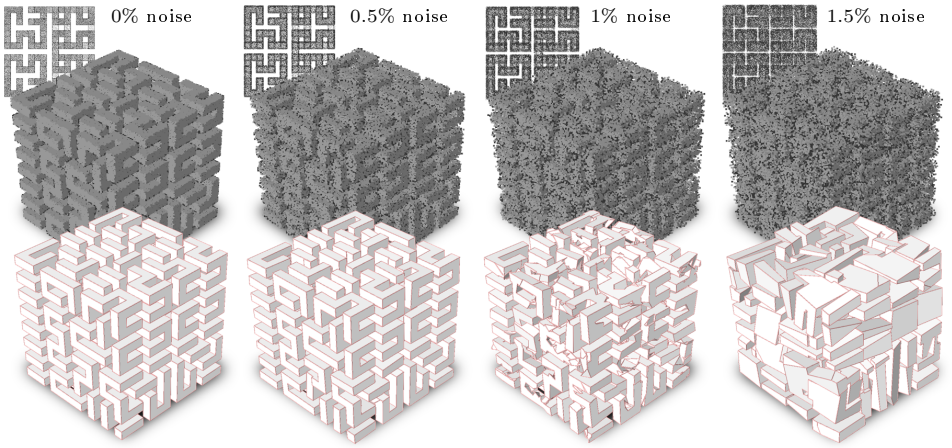


Figure 7.8: Robustness to noise. Our algorithm is robust to noise as long as planar shapes can be decently extracted from input points. At 1% noise, some shapes become missing or inaccurately detected, leading to an overly complex output mesh. At 1.5%, the poorly extracted shapes do not allow us to capture the structure of the cube anymore. Note that normals have been recomputed for each degraded point cloud. Model: Hilbert Cube.

	Meeting room	Full thing	Tower of Pi	Temple	Church
# input points	3.07M	1.38M	2.9M	621K	31.1M
# planar shapes	1,655	1,648	10.8K	69	1,188
#output facets	1,491	1,807	12,051	95	394
Subdivision	4	4	8	-	2
kinetic partitioning (sec)	197	443	2,599	11	310
surface extraction (sec)	84	49	154	10	717
memory peak (MB)	642	552	4,224	69	325

Table 7.2: Performances of our algorithm on various models.

7.2.2.3 Performances

Table 7.2 presents the performances of our algorithm in terms of running time and memory consumption from various models. Kinetic partitioning

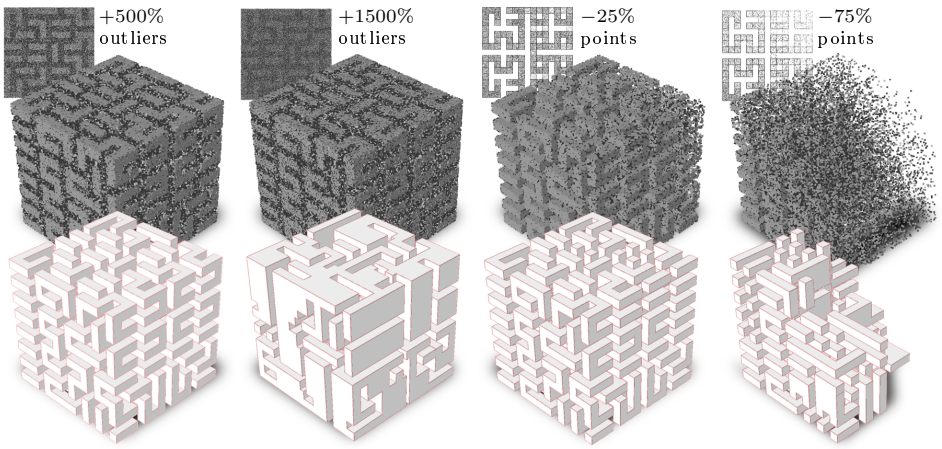


Figure 7.9: Robustness to outliers and heterogeneous sampling. Our algorithm returns an accuracte model when adding 500% of outliers in the object bounding box, but starts compacting the structure around 1500% of added outliers. In the two right examples, the point clouds have been progressively subsampled from the bottom left to the top right of the cube in a linear manner. Our algorithm is resilient to such heterogeneous sampling as long as shapes can be retreived in the low density of points.

is typically the most time-consuming step. In particular, around 90% of computing efforts focuses on the processing of the priority queue. The total number of collisions depends on multiple factors which include the number of initial convex polygons, their number of vertices, their mutual positioning within the bounding box as well as parameter K. For instance, $53K$ collisions are processed for the 75 shapes of the HAND model shown in Figure 7.13. The most frequent collision cases are b and c with an occurrence of 51% and 30% respectively. Case a is the most time-consuming update, because of the insertion of several sliding vertices.

The costly operation for the surface extraction step is the computation of the voting function $d_i(p, x_i)$ which must be performed for each inlier point. The shape detection step, which is not a contribution of our work, typically requires few seconds for processing several millions input points. In terms of scalability, our algorithm can handle dozens of thousands of planar shapes on a standard computer without parallelization schemes.

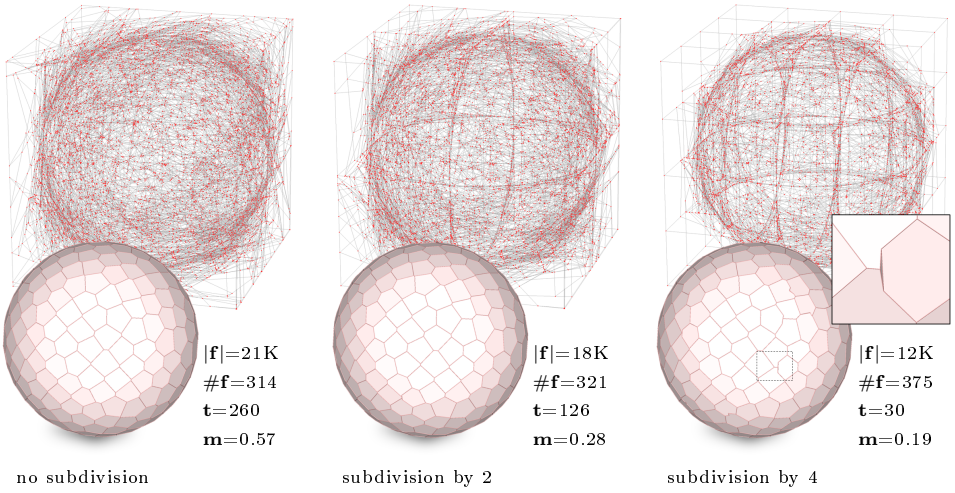


Figure 7.10: Spatial subdivision: efficiency vs. quality. Our algorithm produces an ideal mesh with 314 polygonal facets from 314 planar shapes approximating a sphere (left). Subdividing the bounding box into blocks reduces running time and memory consumption, and produces simplified polyhedra partitions (top). This option may degrade the quality of the output meshes (bottom) with typically the presence of extra facets at the block borders (see closeup). $|\mathbf{f}|$, $\#\mathbf{f}$, $\mathbf{t}$ and $\mathbf{m}$ correspond to the number of facets in the polyhedral partition, the number of polygonal facets in the output mesh, running time (in sec) and memory peak (in GB) respectively.

7.2.2.4 Impact of spatial subdivision

Figure 7.10 shows the impact of the spatial subdivision scheme in terms of performances and quality. We saw in the previous chapter that the decomposition of an initial problem into multiple subproblems of lower complexity decreases the complexity of the partitioning algorithm in terms of time and memory consumptions. However, this may affect the quality of the output surface with typically the presence of extra facets at the borders of blocks.

Therefore, the dimensions of the subdivision grid, which are left to the user, must result from a compromise between efficiency and quality of the surface approximation. The choice of an appropriate grid depends, of course, on the complexity of the initial problem.

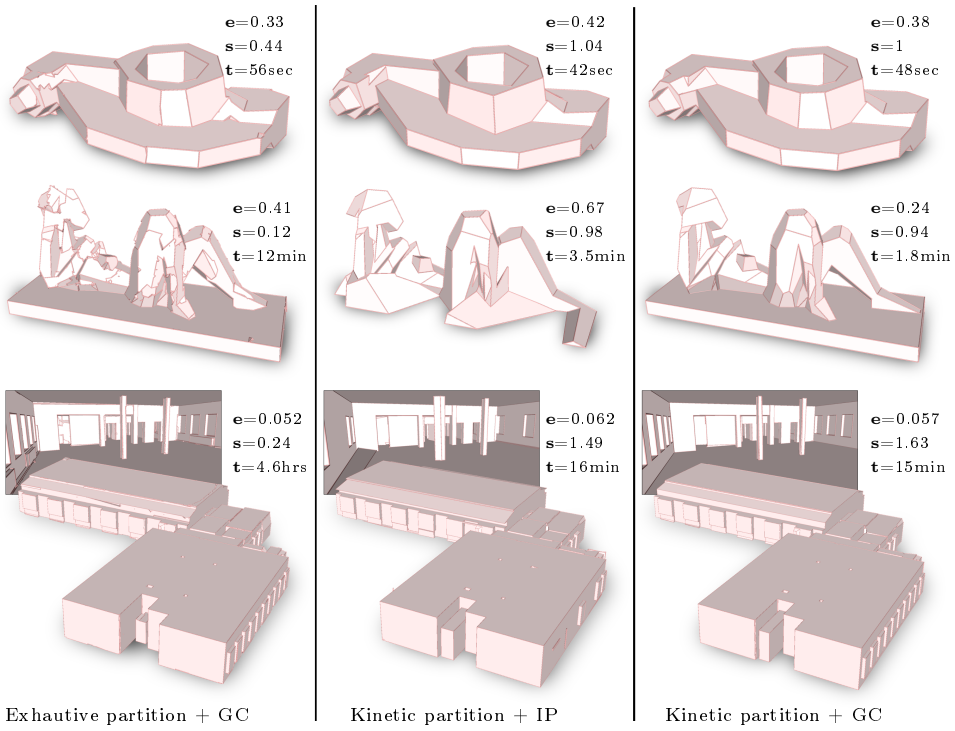


Figure 7.11: Ablation study. Combining exhaustive partitions with our graph-cut (GC) solver typically produces accurate models resulting from the exploration of a large solution space. However, this option is not time-efficient and gives models with a low simplicity score illustrated by numerous visual artifacts (left). Plugging the IP solver on kinetic partitions is a more efficient option, but output models are weakly accurate (middle). Our framework offers the best performances and the best compromise between accuracy and output simplicity (right). Models from Figure 6.6.

7.2.3 Comparisons

7.2.3.1 Ablation study

We evaluated the impact of the kinetic partitioning and the surface extraction modules with an ablation study.

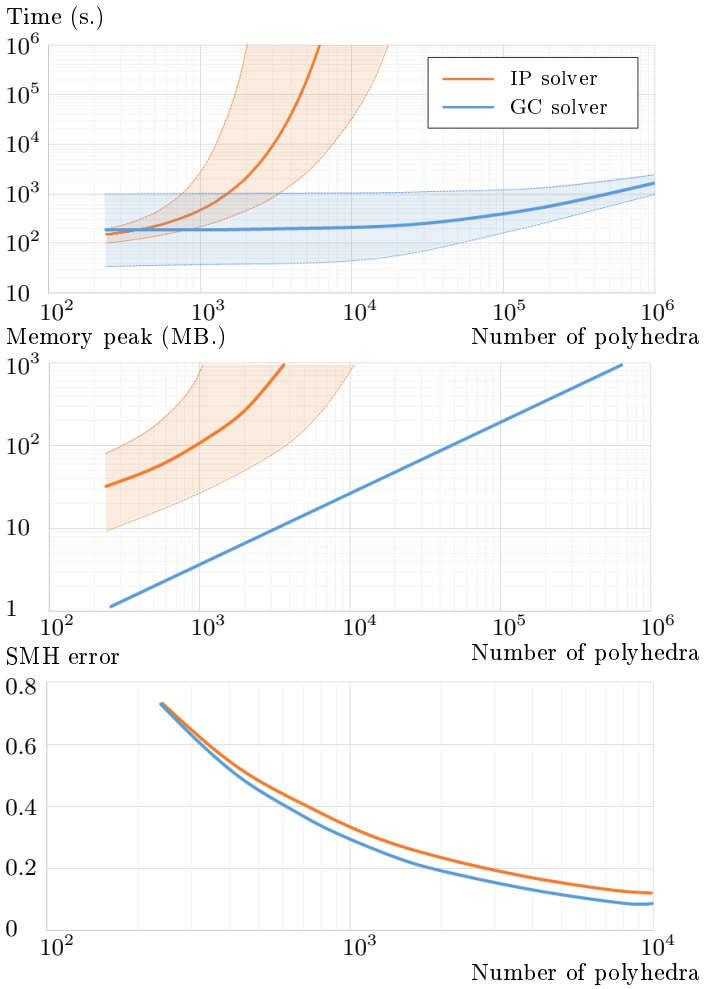


Figure 7.12: Ablation study. When partitions contain less than one thousand polyhedra, our graph-cut (GC) formulation for surface extraction and the integer programming (IP) solver of [NW17] have similar running times and geometric errors (bottom row). However, the IP solver consumes more memory and can hardly digest partitions with more than ten thousands polyhedra. Moreover, the IP solver is less stable as running times and memory peaks can unpredictably vary (see light orange bands in bottom left/middle graphs). In contrast, the variation in time observed for our GC formulation only depends on the number of input points, the lower (respectively upper) bound of the light blue band corresponding to the model with the lowest (resp. highest) number of points. Statistics were drawn from a collection of seven different models with input point sizes ranging from 100K to 5M.

This paragraph is a follow-up to section 6.3.1. Indeed, in the previous chapter, we already compared the performances of exhaustive and kinetic partitionings. Figure 6.7 showed that kinetic partitioning produces more compact sets of polyhedra in a more time- and memory-efficient manner than exhaustive partitioning. In particular, memory consumption is reduced by more than one order of magnitude from 150 planar shapes. Kinetic partitions are also significantly lighter with a reduction of polyhedra by a factor close to the number of shapes when parameter K is fixed to 1. Figure 6.6 illustrated this gap of performances with visual results obtained from 73, 146 and 749 planar shapes.

Here, we measure the efficiency of our graph-cut (GC) formulation against the integer programming (IP) problem proposed in Polyfit [NW17] on kinetic partitions. Running time, memory consumption and MSH error of output models in function of the number of polyhedra in the partition are plotted in Figure 7.12. While MSH errors are similar for small partitions, our graph-cut formulation is significantly more scalable and stable. The solution based on integer programming typically fails to deliver results under reasonable time when partitions contain more than ten thousand polyhedra. Moreover, the convergence time of the branch-and-bound optimization run by the IP solver [GO19] is hardly predictable: small variations on the energy parameters can increase running times by more than three orders of magnitude for a given partition. This solver is also very memory consuming. In contrast, our solver always digests millions of polyhedra in a few minutes, at most. Running times only depend on the number of polyhedra and the size of the point cloud, and memory consumption evolves linearly at a rate of 2KB per polyhedron.

Finally, we measured the quality of output models obtained from different combinations of partitioning schemes and surface extraction solvers in Figure 7.11. The combination of kinetic partitioning with our graph-cut solver delivers the best results in terms of output simplicity and performances, especially when more than one hundred shapes are handled. On the other hand, using exhaustive partitions with our graph-cut solver allows to strongly extend the solution space: this typically produces more accurate models, but at the expense of output simplicity, performance and scalability. Plugging the IP solver on kinetic partitions is an efficient option. However, the IP solver operates on restricted solution spaces in which candidate facets cannot

lie on the object bounding box. This problem both prevents us from using spatial subdivision schemes for increasing scalability, and strongly decreases geometric accuracy of output models in case of occlusions located on the object borders (see the missing ground in the CAPRON model).

Nonetheless, this ablation study highlights the fact that the gain of performances in our solution mainly comes from kinetic partitioning. Note that running the IP solver on exhaustive partitions was not considered in this study, since this solution does not allow to process more than one hundred shapes.

7.2.3.2 Surface reconstruction methods

We compared our algorithm with Polyfit [NW17] and Chauve's method [CLP10], which are arguably the two most robust existing methods in the field. To fairly compare the reconstruction mechanisms, we used the same configuration of planar shapes for all methods. In particular, we deactivated the shape completion module in Chauve's method. Figure 7.13 shows the results obtained on HAND from 75 planar shapes. Because Polyfit and Chauve's method naively slice the object bounding box by the planar shapes, they produce overly-dense partitions. For example, their exhaustive partition for HAND is composed of 30K cells and 91K facets, whereas our partition contains 0.7K cells and 3.5K facets only. Building and extracting output surface from a much lighter partition thus becomes faster and less memory-consuming. Our algorithm also outperforms Polyfit and Chauve's method in terms of fidelity and simplicity. Our surface extraction is more efficient in balancing between fidelity and simplicity. The main reason relies on the use of a simpler data term where only inlier points are taken into account to measure the faithfulness. This technical choice offers more robustness to imperfect data than the visibility criterion of Chauve's method and more stability than the three-term energy of Polyfit.

Graphs presented in Figure 7.14 compare the scalability of the methods. Polyfit may require days of computing when more than one hundred planar shapes are handled whereas Chauve's method exceeds 32GB memory for processing slightly more than two hundred shapes. In particular, the former exploits a time-consuming integer programming solver while the latter suffers from a memory-consuming visibility estimation. Conversely, our algorithm with no subdivision scheme can process one thousand planar shapes under

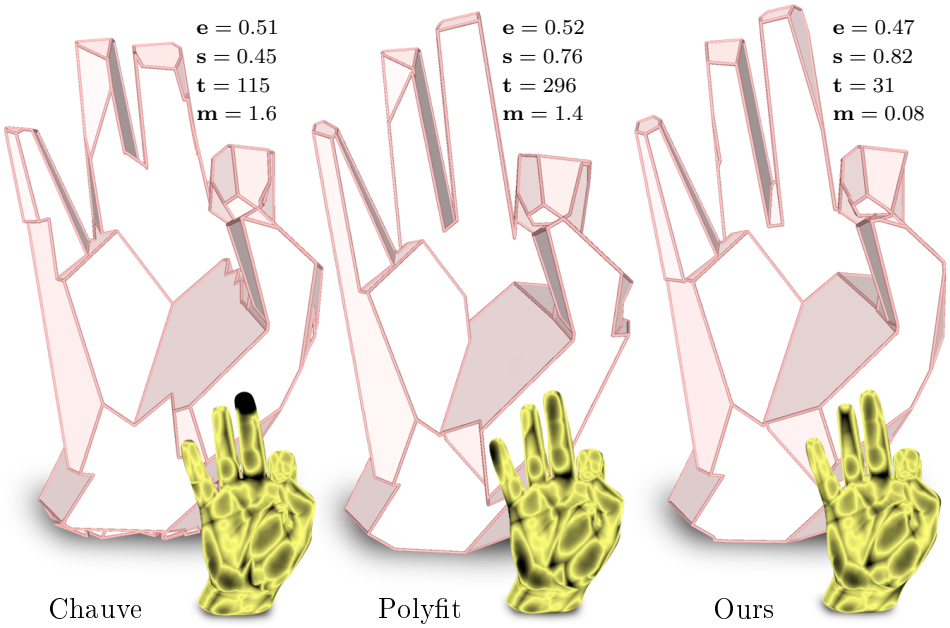


Figure 7.13: Comparisons with surface reconstruction methods. Starting from the same 75 planar shapes, Chauve's method [CLP10] and Polyfit [NW17] are both more time- and memory-consuming than our algorithm while delivering less concise polygonal meshes, both less accurate and less compact. **e**, **s**, **t** and **m** corresponds to the symmetric mean Hausdorff error (in % of the bounding box diagonal), simplicity, running time (in sec) and memory peak (in GB) respectively. The colored point clouds show the error distribution (yellow=0, black≥ 1% of the bounding box diagonal).

more reasonable processing times and without exceeding 32GB memory.

Table 7.3, provides a quantitative comparison between our algorithm, Polyfit and Chauve's method, illustrated by Figures Figures 7.16 and 7.17. Different criteria assess the quality of the output surface, but also the performances of the three algorithms. In our experiments, we considered a set of 42 segmented point clouds, that differ in terms of size, contents and acquisition constraints. More precisely, these point clouds may represent freeform (F) or structured objects (S), but also an indoor (I) or a urban scene (U). They may stem from CAD models, Laser scans, or MVS techniques. We split our dataset into three subsets, namely *simple*, *intermediate* or *advanced* models depending on the number of detected planar shapes in the point clouds: less than 100 shapes for simple models, between 100 and 500 for intermediate

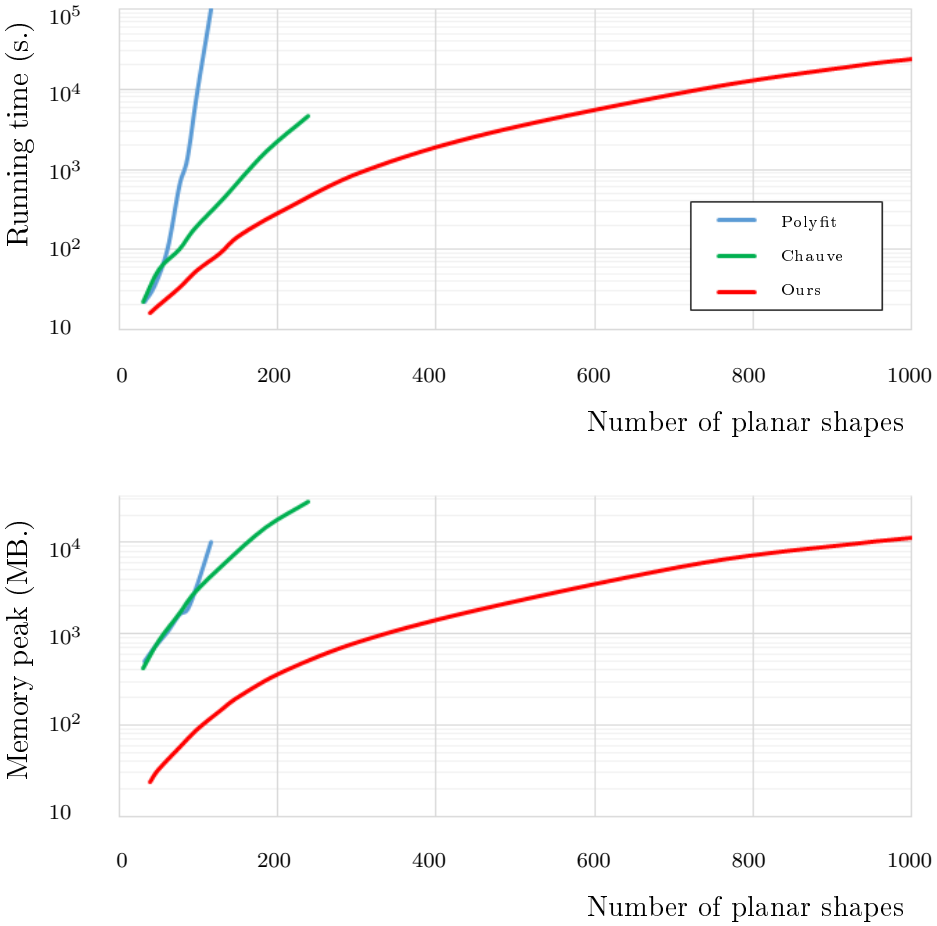


Figure 7.14: Performances of surface reconstruction methods. Polyfit [NW17] requires days of computing for assembling one hundred shapes, whereas Chauve's method [CLP10] exceeds 32GB memory for slightly more than two hundred shapes. Our algorithm is one order magnitude more scalable than these two methods. Tests have been performed on the Hand model (Fig.7.13) without using subdivision schemes.

models (approximatively), and more than 500 for advanced ones.

However, our comparison with Polyfit and Chauve's method only focuses on simple models, since they correspond to the processing capacities of these techniques. The algorithm of Chauve *et al* may process a few models of intermediate complexity: this is illustrated by Table 7.4, and Figure 7.18. We observe that, overall, our algorithm returns simpler surfaces, with a similar or lower geometric error than our competitors. As exhaustive methods

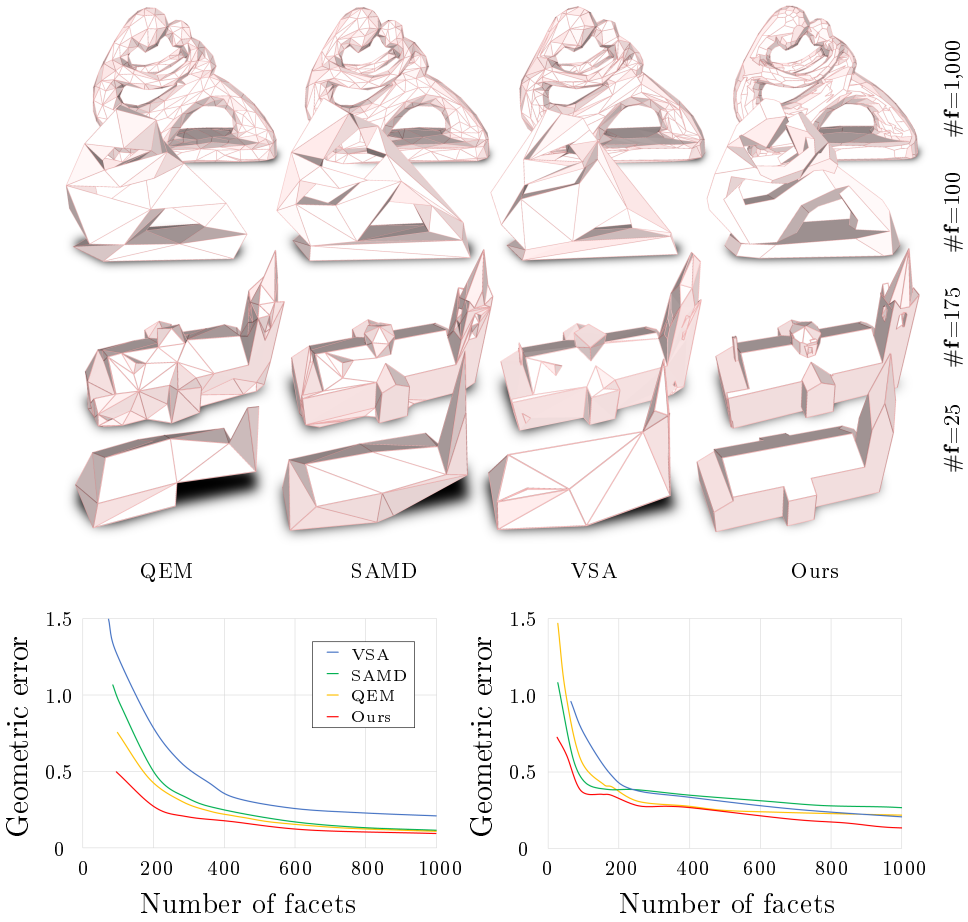


Figure 7.15: Comparisons with surface approximation methods. The left curves, that measure the geometric error in function of the number of facets of the output mesh, show that our algorithm outclasses simplification methods for both a freeform object (Fertility, top) and a nearly piecewise planar structure (Lans, bottom). QEM [GH97] cannot output large meaningful facets whereas SAMD [SLA15] and VSA [CSAD04] fail to correct the geometric inaccuracies contained in the dense triangle mesh. For each row, the four meshes contain the same number of facets #**f**. The geometric error is measured as the symmetric mean Hausdorff distance between input points and output mesh.

benefit from an extended solution space, they sometimes return meshes with lower geometric errors than our technique, but these meshes are also more complex. In addition to this, we observe that our algorithm is characterized by similar or faster running times than other algorithms, and a lower

memory consumption. We plan to release all models on a public repository, including our segmented point clouds, our results, and those generated by surface reconstruction methods.

7.2.3.3 Surface approximation pipelines

We compared our algorithm with the surface approximation methods QEM [GH97], SAMD [SLA15] and VSA [CSAD04]. Because these methods operate from meshes, we first reconstructed a dense triangle mesh from input points using the Screened Poisson (SP) algorithm [KH13]. Both a freeform object (FERTILITY) and a nearly piecewise-planar structure (LANS) were used for this comparison. As shown in Figure 7.15, polyhedral meshes produced by our algorithm are geometrically more accurate than those returned by these three methods at a similar mesh complexity, i.e. with the same number of output facets. The accuracy gain is particularly high at low mesh complexity where approximation methods cannot capture correctly the structure of the object and tend to shrink the output surface. By operating directly from input points, our algorithm does not depend on an intermediate dense mesh reconstruction step in which geometric and topological errors are likely to occur. Moreover, the large polygonal facets returned by our algorithm approximate the object more effectively than the triangle facets returned by edge contraction (QEM and SAMD) or constrained Delaunay triangulation built from a primitive connectivity graph (VSA). In terms of performance, the approximation pipelines have similar processing times and memory consumption to our algorithm when producing low complexity meshes. For instance, the four methods process the LANS model with 25 output facets (see bottom row in Figure 7.15) in approximately half a minute. High complexity meshes, i.e. with more than 200 facets, are produced more quickly by the approximation pipelines than by our algorithm.

Tables 7.3, 7.4 and 7.5 propose a quantitative comparison between our algorithm and surface approximation pipelines on the full dataset of 42 models mentioned before. A qualitative comparison of all algorithms, on the same models, is provided by Figures 7.16 to 7.21. Results confirm that our algorithm competes with surface approximation pipelines based on QEM or VSA, by relying on quantitative indicators. Admittedly, surface approximation pipelines may complete in a shorter time, especially when the input point cloud contains a lot of planar shapes. However, for an equal or lower

number of facets, our technique almost always returns meshes with the lowest geometric error.

All models listed and compared in Tables 7.3, 7.4 and 7.5 will be made public to the Computer Graphics community in near future.

7.2.4 Limitations

Our work focuses on assembling planar shapes, not on detecting them. We assume planar shapes can be accurately detected from input points by standard algorithms.

However, this assumption might be wrong in presence of data highly corrupted by noise and severe occlusions. For such cases, configurations of planar shapes returned by existing algorithms are typically inaccurate and incomplete. Our kinetic algorithm can only correct configurations where missing planar shapes are parts of a repetitive structure as illustrated in Figure 6.4.

		Barn MVS	Building A	Building B	Building C	Bunny	Chair	Cottage	Couch
Type		U	U	U	U	F	S	U	S
Origin		MVS	MVS	MVS	MVS	Laser	Laser	MVS	Laser
#p		619K	101K	73K	577K	146K	756K	143K	911K
#s		95	20	34	32	98	23	23	21
KSR	e_A	**0.231**	**0.951**	**0.605**	**0.372**	**0.454**	**0.557**	**0.401**	**0.560**
	e_S	**0.179**	1.522	**0.735**	0.344	**0.540**	**0.580**	**0.471**	**1.681**
	s	**2.5**	**0.690**	1.478	**0.941**	**0.856**	**1**	**0.821**	**0.84**
	#f	38	29	23	34	111	23	28	25
	t	28.9	3.4	2.5	18.4	53.3	25	4.2	25.1
	m	53	10	13	15	97	34	15.7	11.6
Polyfit	e_A	-	1.347	2.097	0.461	-	1.447	1.491	6.385
	e_S	-	**1.170**	1.385	0.343	-	1.089	1.259	3.325
	s	-	1.429	**1.36**	1.067	-	1.438	1.278	1.235
	#f	-	14	25	30	-	16	18	17
	t	-	2.1	2.5	10.2	-	16.9	5.3	18.8
	m	-	144	169	252	-	241	177	299
Chauve *et al*	e_A	1.042	9.585	2.626	0.407	0.545	1.422	1.429	3.931
	e_S	0.772	5.172	1.704	**0.308**	0.555	1.082	0.823	2.137
	s	3.8	0.625	0.330	0.571	0.375	0.92	1.278	1.5
	#f	25	32	103	56	261	25	18	14
	t	131.2	0.4	1.9	5.5	94	9.7	1.5	6.7
	m	3189	10	103	125	2924	501	56	308
SP + QEM	e_A	2.815	4.081	6.344	2.873	1.435	4.287	2.373	2.922
	e_S	1.575	2.482	3.856	1.934	1.069	3.000	1.513	1.971
	#f	39	28	22	26	110	24	27	25
	t	36	17.4	14.2	21.4	21	70.1	22.7	22.8
	m	232	162	111	458	195	495	156	406
SP + VSA	e_A	0.752	4.222	2.673	1.593	1.002	1.906	1.181	1.762
	e_S	0.538	3.107	1.921	1.316	1.031	1.664	0.958	1.806
	#f	717	213	157	947	175	420	528	113
	t	45.3	27.5	15.1	24.3	27.1	96.7	26.4	27.1
	m	675	428	347	480	352	1572	473	556

		Fertility-Coarse	Foam box	Hand	Lans-Coarse	Rocker arm	Rooms A	Rooms B	Temple
Type		F	S	F	U	S	I	I	S
Origin		Laser	Laser	Laser	Laser	Laser	Laser	Laser	MVS
#p		242K	382K	369K	1.22M	733K	186K	176K	621K
#s		75	61	75	24	73	48	28	69
KSR	e_A	0.350	0.247	**0.423**	0.491	**0.257**	0.677	0.553	**0.381**
	e_S	0.498	0.291	**0.480**	**0.726**	**0.382**	0.603	0.538	**0.481**
	s	**0.789**	**0.968**	**0.815**	**0.96**	**1**	**1.043**	**1.077**	0.726
	#f	95	63	92	25	73	46	26	95
	t	25	15	31	33.4	43.4	7.9	4.8	21.6
	m	61.2	29	76.6	11.5	53.8	19.1	8.8	48.1
Polyfit	e_A	0.388	0.221	0.460	3.977	0.361	**0.590**	0.680	0.548
	e_S	0.500	0.297	0.537	2.592	0.524	**0.556**	0.572	0.683
	s	0.5	**0.968**	0.765	0.923	0.507	0.941	1.474	**0.767**
	#f	150	63	98	26	144	51	19	90
	t	1070	672	296	26.6	9343	4.5	3.1	1943
	m	1919	1748	1600	331	1896	221	135	900
Chauve *et al*	e_A	**0.328**	**0.207**	0.462	**0.421**	0.357	0.790	**0.526**	0.459
	e_S	**0.407**	**0.261**	0.488	1.195	0.411	0.671	**0.465**	0.911
	s	0.142	0.570	0.399	0.667	0.361	0.384	0.519	0.099
	#f	528	107	188	36	202	125	54	700
	t	53.1	31.5	115	4.4	93.6	8.9	1.7	76.7
	m	1037	1131	1400	96	2141	196	56	1673

SP + QEM	$\mathbf{e_A}$	0.722	2.969	0.724	2.814	2.124	2.847	2.871	2.127
	$\mathbf{e_S}$	0.756	1.933	0.777	1.831	1.509	1.984	1.645	1.498
	#f	100	62	92	25	72	46	26	92
	t	38.5	55	29.1	31.6	38.1	34.7	29.9	43.5
	m	289	518	241	311	142	377	399	453
SP + VSA	$\mathbf{e_A}$	1.724	1.404	1.058	2.894	1.201	2.066	0.911	0.894
	$\mathbf{e_S}$	1.287	0.960	0.918	2.355	1.091	1.422	0.704	0.806
	#f	93	682	127	31	83	568	123	270
	t	28.8	78.8	39.6	31.8	54.4	41.2	40.2	69.3
	m	361	1341	740	311	142	901	758	1013

Table 7.3: Quantitative comparison between surface reconstruction and surface approximation algorithms on simple models. Models Building A, Building B, Building C, Rooms A, Rooms B, Foam box, Chair and Couch are from Nan *et al* [NW17] and correspond to the Figures 4-(b) to 4-(i) of their paper. $\mathbf{e_A}$ stands for the asymmetric geometric error between the input point cloud and the output model (point cloud to mesh), and $\mathbf{e_S}$ for the symmetric geometric error. **s** represents the simplicity index of the output surface, **#f** its number of facets. **t** indicates the running time of all methods (in seconds) and **m**, the memory peak (in megabytes). Polyfit was unable to process models Barn MVS and Bunny after several hours of execution. Visual results are listed in Figures 7.16 and 7.17.

		Barn Laser	Building Block	Capron	Hilbert Cube	Hilbert Cube (N 0.5)	Hilbert Cube (N 1)	Hilbert Cube (N 1.5)	Hilbert Cube (O 500)
Type		U	U	F	S	S	S	S	S
Origin		Laser	MVS	MVS	CAD	CAD	CAD	CAD	CAD
#p		2.1M	793K	168K	144K	144K	144K	144K	864K
#s		222	160	146	986	1135	2601	480	1003
KSR	$\mathbf{e_A}$	**0.133**	**0.232**	0.282	**0**	**0.030**	**0.122**	**0.741**	**0.009**
	$\mathbf{e_S}$	**0.128**	**0.222**	**0.290**	**0.124**	**0.140**	**0.229**	**0.929**	**0.128**
	#f	142	142	156	986	997	2087	894	986
	t	93.5	107.5	106.8	16.1	16.8	121.2	21.4	18.2
	m	54	147	194	105	66	451	62	59
Chauve *et al*	$\mathbf{e_A}$	0.465	1.115	**0.223**	-	-	-	-	-
	$\mathbf{e_S}$	0.627	0.419	0.466	-	-	-	-	-
	#f	814	589	10k	-	-	-	-	-
	t	2475	333.4	555.7	-	-	-	-	-
	m	> 32 GB	5361	6080	-	-	-	-	-
SP + QEM	$\mathbf{e_A}$	0.375	1.094	0.855	1.501	1.583	0.858	1.382	3.794
	$\mathbf{e_S}$	0.339	0.690	0.587	1.044	1.060	0.640	1.156	2.420
	#f	143	143	155	986	998	2088	894	978
	t	28.8	27.8	34.5	26.8	29	27.7	27.7	161.4
	m	307	391	98	155	179	367	350	1427
SP + VSA	$\mathbf{e_A}$	0.388	0.566	0.738	4.884	2.173	2.002	2.962	3.459
	$\mathbf{e_S}$	0.332	0.454	0.5	2.594	1.288	1.252	1.807	2.286
	#f	501	158	712	2486	2737	2049	1938	55K
	t	33.8	35.3	44	36	40	36.5	35.6	210.3
	m	579	551	608	507	577	617	666	4617

		Hilbert Cube (O 1500)	Hilbert Cube (D -75)	Hilbert Cube (D -25)	Horse	Ignatius	Lans-Fine	M60	Sphere
Type		S	S	S	F	F	U	S	F
Origin		CAD	CAD	CAD	MVS	MVS	Laser	MVS	CAD
#p		2.3M	36K	108K	788K	1.4M	1.2M	2.8M	100K
#s		1184	531	965	274	294	306	362	304
KSR	e_A	**0.537**	4.755	**0.021**	**0.249**	**0.212**	0.112	**0.442**	**0.054**
	e_S	**0.448**	2.513	**0.135**	**0.237**	0.288	**0.355**	**0.348**	**0.119**
	#f	626	720	1032	351	443	179	471	319
	t	19.9	7.4	13.8	712.8	201.8	576.7	281.1	138.9
	m	50	79	126	1018	372	824	378	256
Chauve *et al*	e_A	-	-	-	-	-	**0.083**	-	**0.054**
	e_S	-	-	-	-	-	0.491	-	0.387
	#f	-	-	-	-	-	3719	-	1256
	t	-	-	-	-	-	6510	-	294
	m	-	-	-	-	-	> 32 GB	-	8616
SP + QEM	e_A	10.89	**2.172**	1.465	0.471	0.330	0.255	0.867	0.174
	e_S	6.073	**1.439**	1.025	0.341	**0.277**	0.386	0.627	0.214
	#f	568	721	1030	350	443	175	470	318
	t	494.4	7.8	22.5	38.4	35.8	31.6	43.3	24.6
	m	4576	137	156	339	422	311	287	198
SP + VSA	e_A	0.815	3.655	5.722	1.208	0.725	0.354	0.533	0.060
	e_S	0.975	2.223	3.096	0.818	0.625	0.383	0.427	0.141
	#f	489K	887	1081	451	468	164	1694	311
	t	806.5	10.1	30.4	52	47.7	37	58.6	40
	m	15571	179	492	825	773	311	772	487

Table 7.4: Quantitative comparison between our algorithm, Chauve's method, and surface approximation pipelines on models of intermediate complexity (between 100 and 500 shapes, in average). For all versions of Hilbert Cube, we consider the ideal, non-perturbed point set as reference. We refer the reader to Figures 7.18 and 7.19 for visual comparisons between these models.

		Asian dragon	Castle	Church	Courthouse	Euler
Type		F	U	I	U	I
Origin		Laser	CAD	Laser	Laser	Laser
#p		3.6M	737K	31.1M	1.9M	2.7M
#s		2712	735	1188	2716	887
KSR	e_A	**0.065**	**0.006**	0.460	**0.134**	**0.069**
	e_S	**0.069**	**0.038**	**0.292**	**0.183**	**0.108**
	#f	3132	711	394	1795	1317
	t	1109	268.8	1040	375.4	354.2
	m	1667	321	324	862	711
SP + QEM	e_A	0.071	0.044	**0.292**	0.155	0.148
	e_S	0.072	0.082	0.345	0.245	0.237
	#f	3132	712	394	1795	1318
	t	27.6	100.5	42.8	44.3	40.5
	m	300	1127	558	352	639
SP + VSA	e_A	0.125	0.098	0.569	0.254	0.242
	e_S	0.108	0.110	0.377	0.262	0.240
	#f	3351	1894	494	1818	4070
	t	49.9	146.1	47.1	67.8	39.7
	m	594	1975	588	876	870

		Fertility-Fine	Full thing	Meeting Room	Navhis	Tower of Pi
Type		F	S	I	I	F
Origin		Laser	CAD	Laser	Laser	CAD
#p		242K	1.4M	3.1M	3.6M	2.9M
#s		812	1648	1655	749	10834
KSR	e_A	**0.061**	**0.064**	**0.198**	**0.061**	**0.020**
	e_S	**0.095**	**0.064**	**0.146**	**0.057**	**0.039**
	#f	998	1807	1491	457	12051
	t	1467	462.1	268.4	962.1	2754
	m	1466	552	642	1569	4224
SP + QEM	e_A	0.082	0.170	0.229	0.167	0.135
	e_S	0.117	0.141	0.204	0.146	0.098
	#f	1000	1806	1490	458	12050
	t	41.3	67.3	65.5	36.4	205.8
	m	361	595	869	646	1422
SP + VSA	e_A	0.183	0.378	0.202	0.192	0.295
	e_S	0.243	0.245	0.160	0.164	0.241
	#f	982	4368	4949	522	27285
	t	30	104	105.8	46.3	365.1
	m	361	1455	1328	692	4176

Table 7.5: Quantitative comparison between our algorithm and surface approximation pipelines on models of high complexity (more than 500 shapes). A qualitative comparison is provided by Figures 7.20 and 7.21.

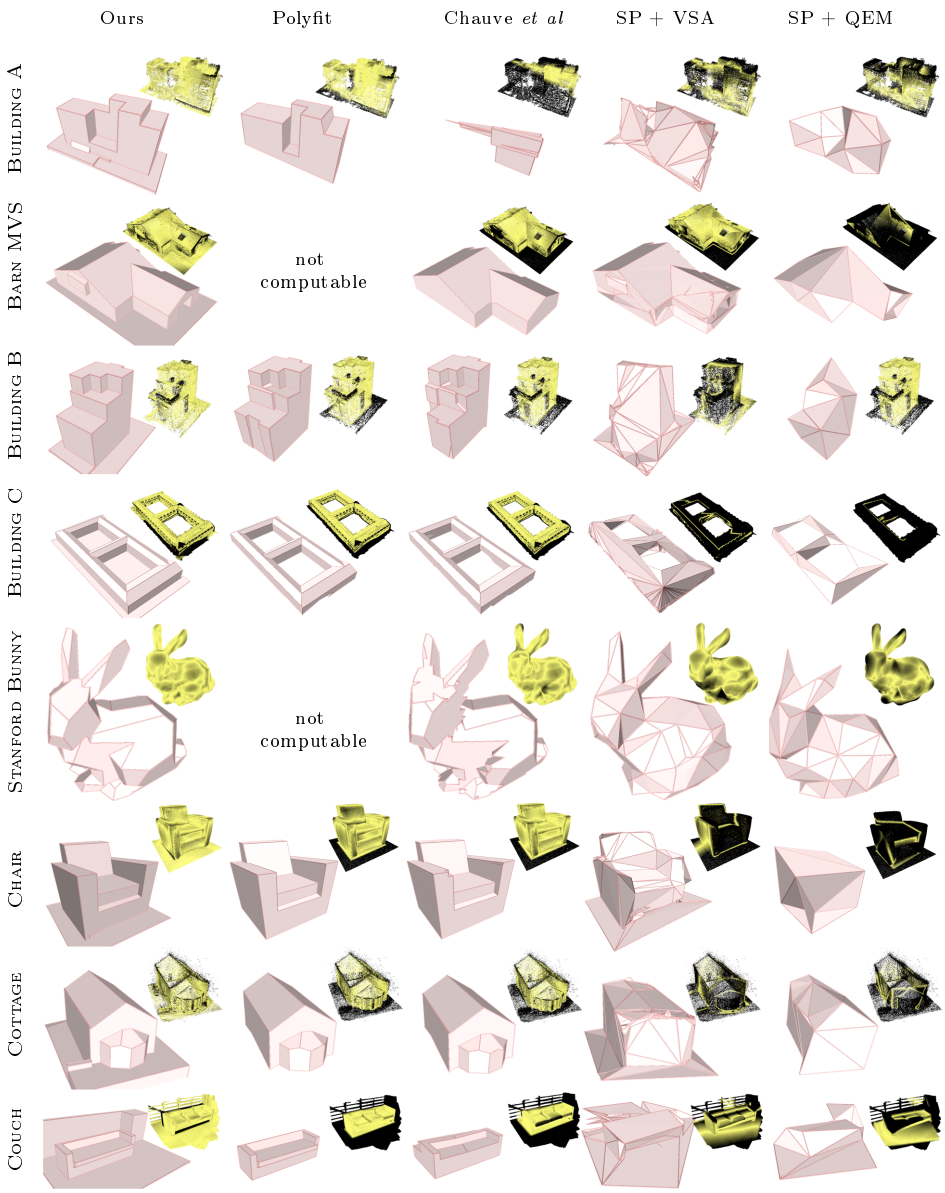


Figure 7.16: Qualitative comparisons on simple models (part 1). The error maps correspond to the Hausdorff error from input points to output models and ranges from 0 (yellow) to ε or higher (black), ε being the fitting tolerance parameter for the detection of planar shapes (fixed similarly for each model).

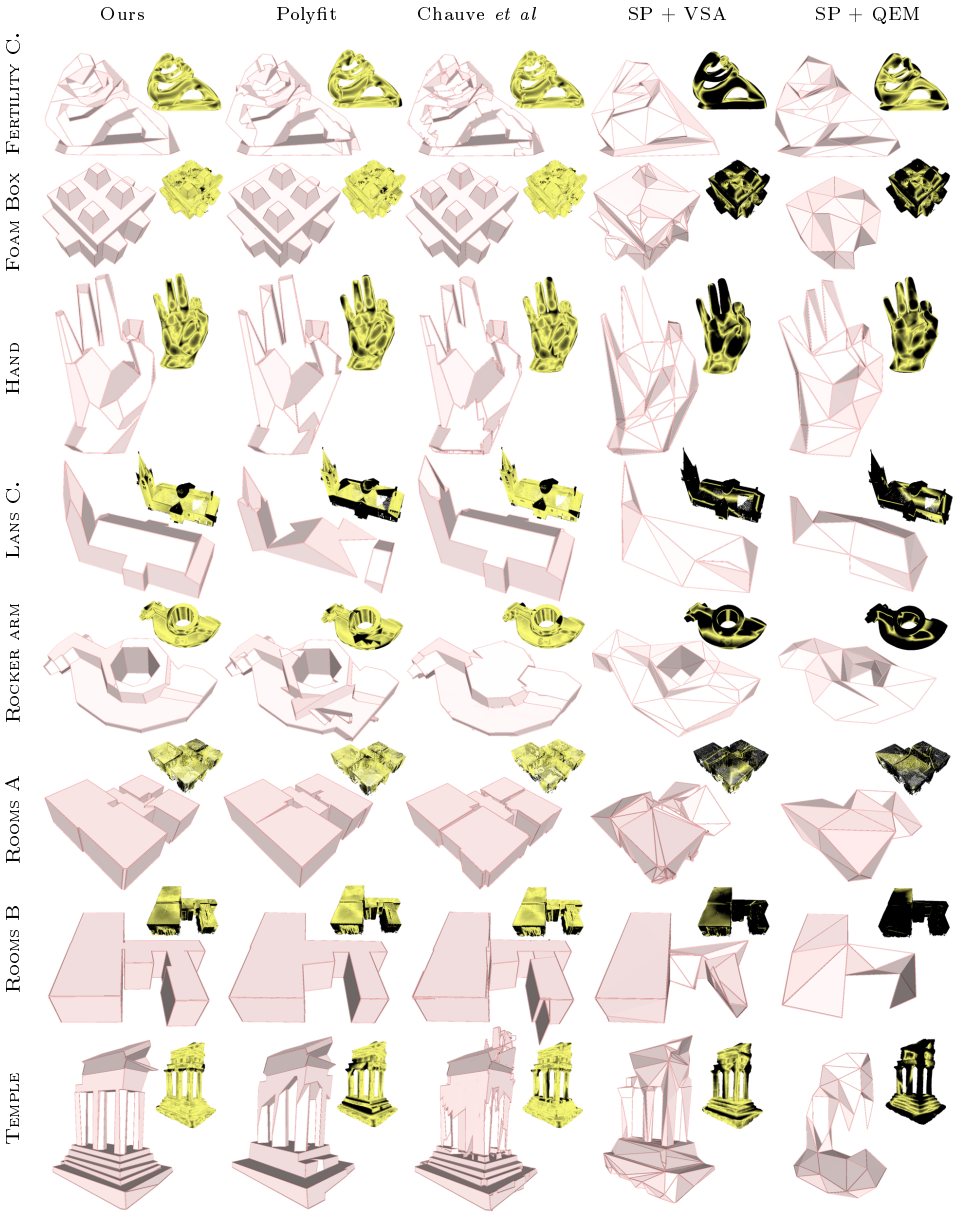


Figure 7.17: Qualitative comparisons on simple models (part 2).

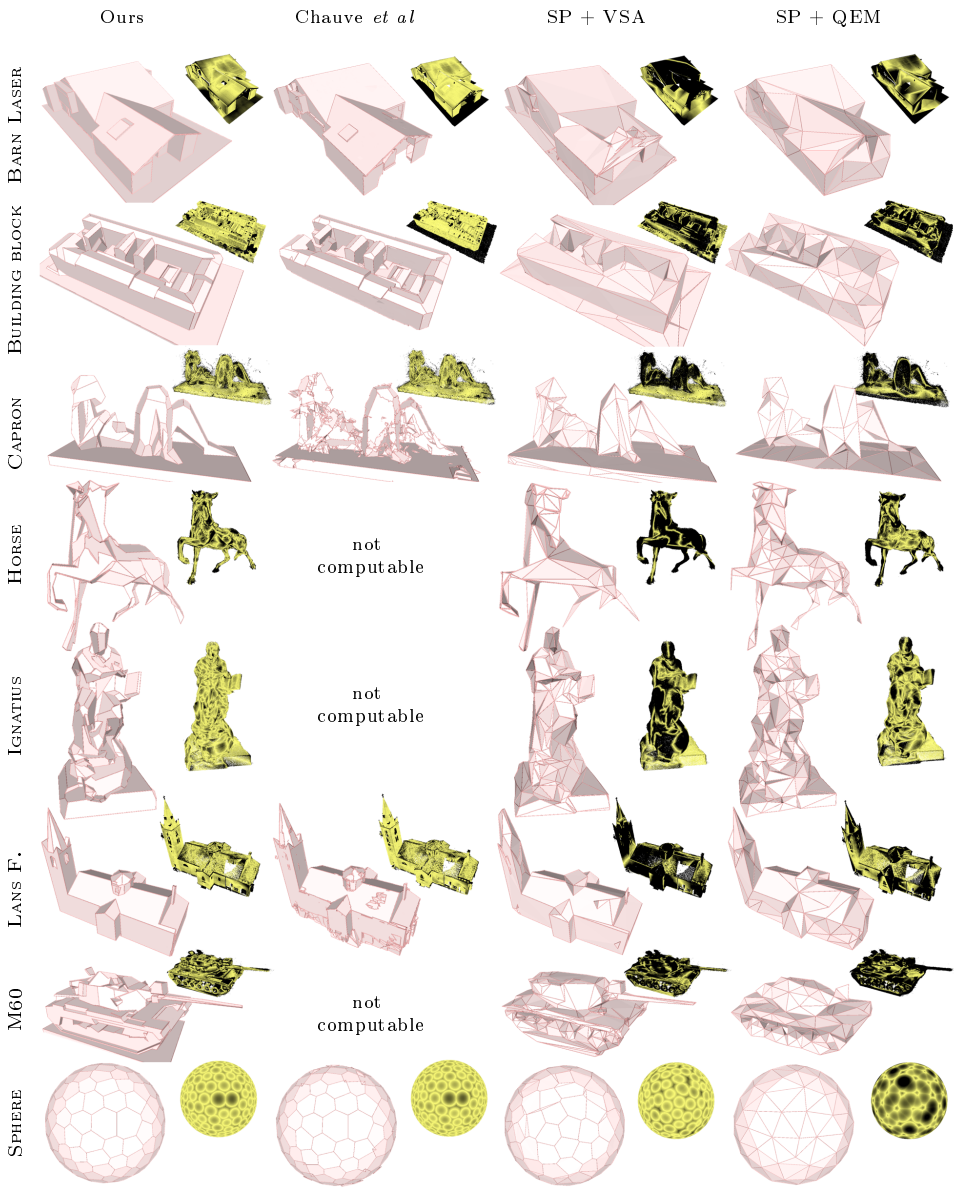


Figure 7.18: Qualitative comparisons on intermediate models (part 1).

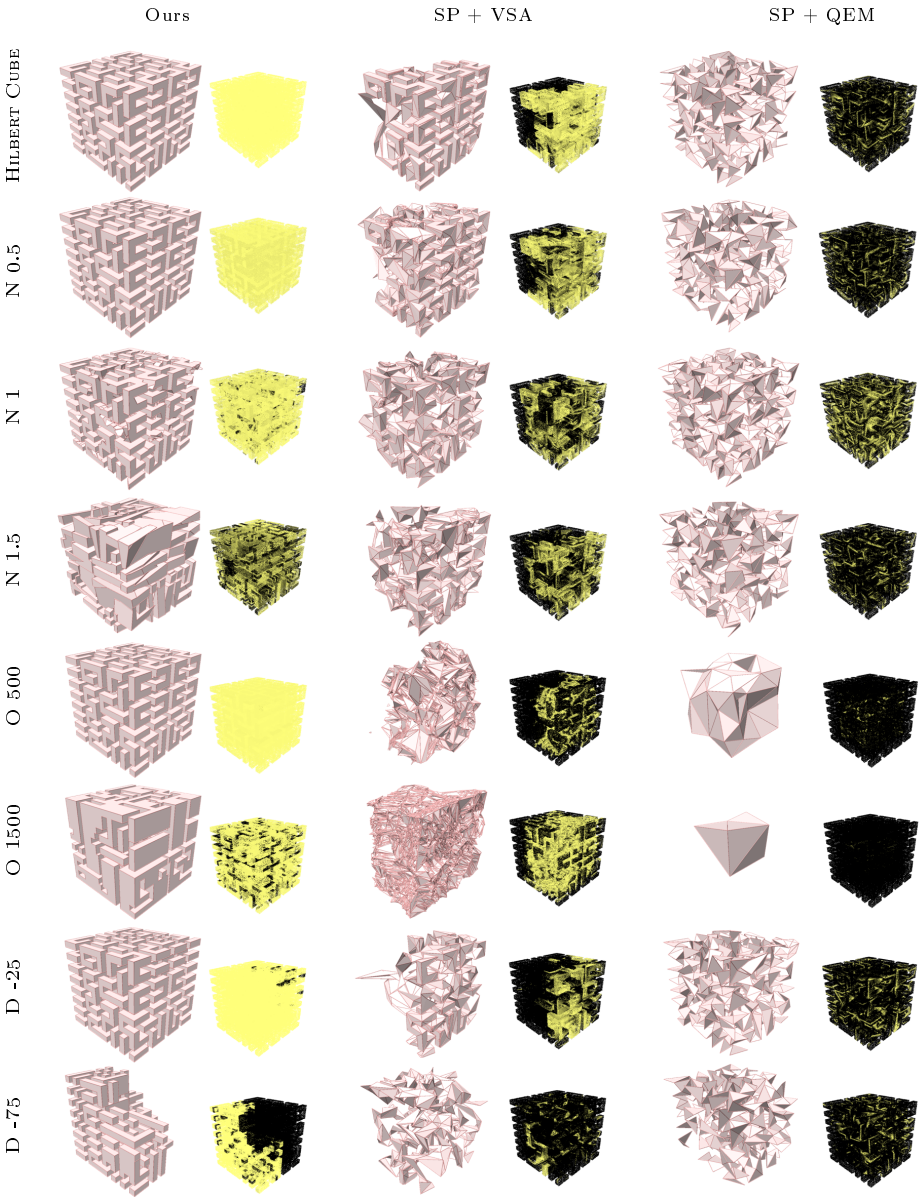


Figure 7.19: Qualitative comparisons on intermediate models (part 2). This figure only considers the model Hilbert Cube and its defect-laden variants, presented in Figures 7.8 and 7.9.

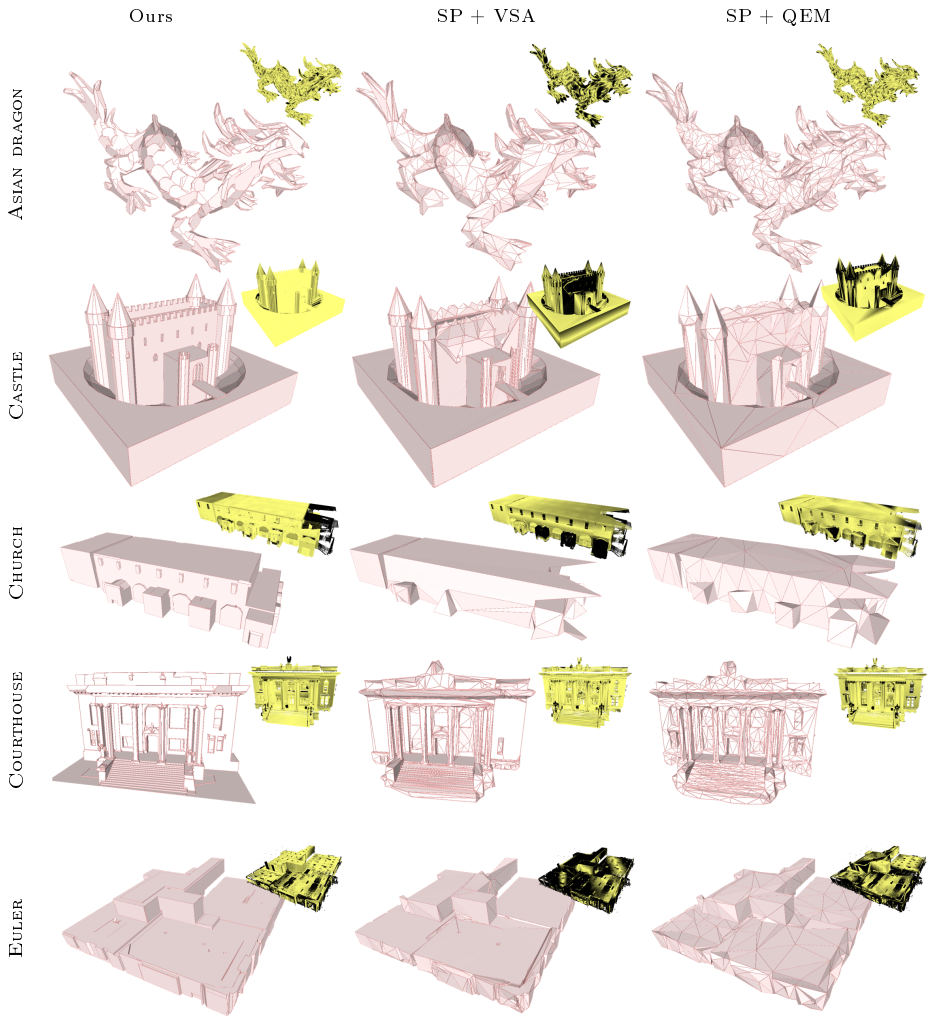


Figure 7.20: Qualitative comparisons on advanced models (part 1).

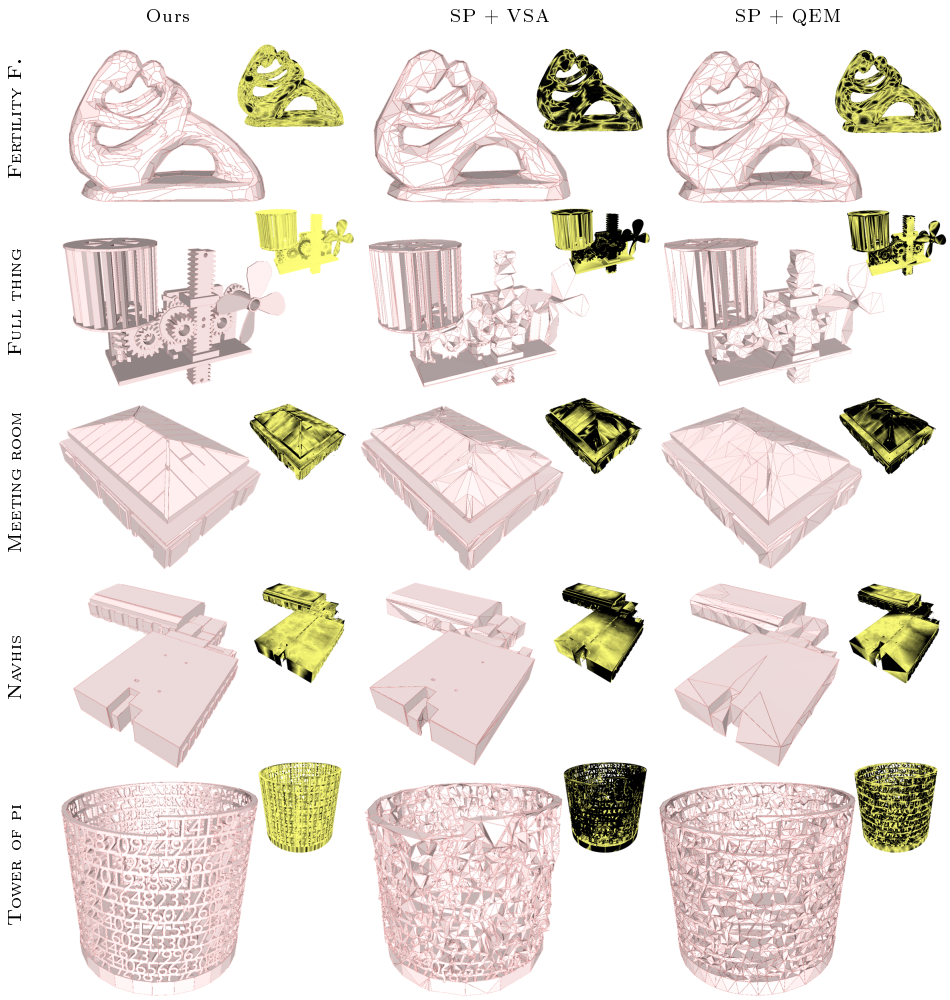


Figure 7.21: Qualitative comparisons on advanced models (part 2).

7.3 Conclusions

This chapter exposed the last part of our object approximation pipeline for images. Starting with a kinetic partition of this image, we implemented a simple object contouring model based on the min-cut/max-flow algorithm, in order to select the polygonal cells that are part of the objects of interest in the image. These objects can therefore be approximated using a small set of edges. In spite of its simplicity, our model shows the applicative potential of kinetic partitions for object approximation in images.

Likewise, by combining the polyhedral partitioning algorithm and a surface extraction model based on graph cuts, we obtain a full pipeline for converting an oriented point cloud into a compact polygonal mesh. Our experiments demonstrate the flexibility and the robustness of our approach, that delivers compact, yet geometrically accurate surfaces. A qualitative and quantitative comparison, conducted on a wide range of datasets, show that our algorithm can compete with existing surface reconstruction methods and traditional approximation pipelines. We carried out an ablation study, evaluating the impact of the kinetic partitioning and surface extraction modules of our processing chain. It highlights, on one hand, the perfomance gain induced by our polyhedral partitioning technique, and on the other hand, the stability and the scalability of our solver compared with another method based on integer programming.

Chapter 8

Application to urban modeling

Urban reconstruction techniques have attracted an increasing attention from the scientific community over the last decade. Applications of such techniques include, for instance, urban planning, natural disaster management, or radio-wave propagation [BSL+15].

Various data sources are considered by such algorithms, such as ground imagery, satellite imagery, or pre-existing GIS data. However, despite high acquisition costs, Lidar point clouds are more accurate and remain a data source commonly used by land surveying offices or civil engineers [MWA+13].

Yet, models generated by current city modeling techniques may be enriched with semantic information, or represented using progressive levels of detail. The CityGML standard [GKNH12] considers four coarse-to-fine levels applicable to airborne datasets, from LOD0 to LOD3.

In this chapter, we present a pipeline that receives as input an unstructured point set describing a large-scale urban scene, and generates as output a LOD2 representation of the scene in compliance with the CityGML standard, meaning that we aim at providing a faithful reconstruction of buildings with tilted roofs. LOD2 models can be used for visualization purposes, visibility analyses, solar radiation estimation, and other tasks LOD1 models cannot be used for. Our method consists of three main steps: classification, building contouring, and building reconstruction, the two last steps being achieved using kinetic data structures. We emphasize the accuracy and the scalability of our method, since it is able to process large datasets with millions of points, dense or sparse, in a few minutes.

8.1 Related works

There exists a vast literature on automatic urban reconstruction techniques, [HK10, RSG+14, MWA+13], demonstrating the deep interest of scientists and industrials for this research topic. Various data sources may be consid-

ered, leading us to draw a first distinction between all existing approaches, based on this criterion. Indeed, some algorithms may specifically address the problem of large-scale urban reconstruction from aerial imagery [ZBKB08, ZWF18], satellite imagery [DL16] or multi-view stereo dense meshes [ZSGH18, VLA15, SLA15]. Some others combine different sources of data to generate urban models with the finest level of detail [KFWM17].

In this work, we focus on the problem of city modeling from Lidar point clouds. Roof height estimation and building reconstruction is often the most valuable information to extract from such data, which is acquired using terrestrial or aerial devices. In particular, nadir or near-nadir acquisitions pose a specific constraint, as facades are missed by scanners. Current state-of-the-art approaches have been extensively reviewed [WPC18] and can be divided in three categories.

Data-driven methods are probably the most popular techniques. These are bottom-up approaches, in which parametric primitives are extracted from the data and assembled to form a reconstructed model. Existing pipelines typically consist of three successive steps: classification, segmentation and geometric modeling. The semantic interpretation of the data, and the clustering of buildings into individual structures may involve statistical arguments [PY09] or discriminative geometric features coupled with energetic formulations [LM11]. However, the models produced by such techniques do not achieve the same level of detail. For instance, the methods proposed in [ZN09] and [Pou13] only reconstruct multi-level flat buildings from airborne point clouds, which is suitable for Manhattan-like districts but less accurate for residential areas. In contrast, the algorithm of [SHT08] considers a binary space partitioning tree to generate LOD2 polyhedral meshes from a point cloud. The one of [ZN10] reaches a similar level of detail by minimizing 2.5D quadric error functions, i.e. taking into account both the surface being reconstructed and its projected boundary. However, due to the projection of the points on a 2D grid, the reconstructed facades show a zig-zagging effect, which might be corrected by the mesh simplification procedure described in [ZN12]. The cell decomposition approach proposed in [KM09] allows to reconstruct buildings with a compact representation, but requires precise building footprints as input. The method of [LM11] also returns persuasive results over large-scale areas, but suffers from the same failing. Recently, deep-learning-based methods have also been developed in a

context of LOD2 urban modeling and achieve very promising results [ZZ18].

Model-driven methods, for their part, represent the opposite, top-down strategy. This family of techniques considers a pre-defined library of template structures (e.g. flat, gable, hip or mansard roofs) that is matched to the input data. The work of [VKH06] offers a first example of model-driven algorithm: elements of the point cloud are first classified as flat or non-flat and a roof topology graph is considered to decompose a complex building into simpler structures. Another example is the work of [HBS13] in which a stochastic approach is used to select the roof templates that best fit the input data. Also requiring building footprints as prior knowledge, the method of [HGSP13] uses RANSAC and supervised machine learning techniques to generate a LOD2 reconstruction of a sparse Lidar point in a model-driven way. However, such approaches may lack of flexibility with respect to the variety of urban landscapes.

Finally, hybrid-driven methods try to take the best of both worlds: parameterized primitives are extracted and assembled with respect to a set of constraints derived from constructive solid geometry [XEV14, LDZPD10].

In the following, we present an algorithm that addresses the problems we exposed before by exploiting powerful computational geometry tools. Given an airborne input point cloud, we design a scalable and data-driven algorithm that generates LOD2 representations of different urban environments as concise polyhedral meshes.

8.2 Pipeline

8.2.1 Input

As depicted by Figure 8.1, our algorithm takes as input a point cloud with oriented normals. Normals can be easily estimated thanks to the acquisition system information. If not provided, then basic mathematical tools like principal component analysis can be used.

8.2.2 Classification

The first step of our pipeline consists in assigning semantic labels to points of the point cloud. Three labels are considered: *ground*, *vegetation*, and *building*.

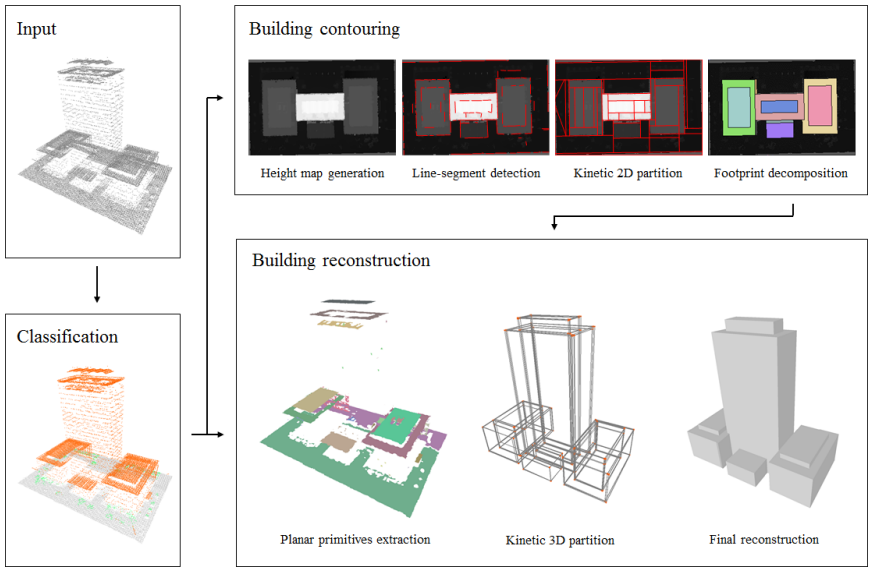


Figure 8.1: Overview of our pipeline. Our method consists of three main steps. We first label points of the Lidar scan as ground, vegetation or roof. Then, we apply a contouring algorithm to the height map, revealing the facades initially absent in the point set. Finally, we extract and propagate planar primitives from the point cloud, dividing the space into polyhedra that are labelled to obtain a 3D reconstruction of buildings.

To this end, we rely on the classification package provided by the CGAL library [GL19]. For each point of the input dataset, this method computes multi-scale geometric features such as elevation, planarity or vertical dispersion for instance. Extra features provided along with the dataset, like the number of returns, are also taken into account. Given a ground truth training set, these features are then used to train a classifier. The default choice for the classifier is a random forest, that constructs several decision trees to assign each point to one of the three aforementioned subsets.

Interactively labelling the data to get a representative training dataset is a tedious work. However, the CGAL library offers the possibility to save and reuse a trained classifier, which is particularly useful for processing urban scenes of similar nature (dense or surburban areas, downtowns, historical centers...).

8.2.3 Building contouring

Facades of buildings are often partially or completely missing in aerial point clouds. To obtain accurate reconstructions of cities from such data, we first need a robust method that detects significant height discontinuities in the classified point cloud.

To this end, we project all points on a horizontal, uniformly sampled grid, in order to generate two kinds of maps: (i) a height map, normalized as a grayscale image, (ii) a probability map, measuring the proportion of projected points labelled as *buildings* in each cell of the grid.

These two maps are then processed by the polygonal partitioning technique detailed in chapter 5. We remind the reader that, given an input image, our algorithm first detects line-segments, which are linear approximations of regions where the image gradient is high and regular. These line-segments propagate across the image, until intersecting each other, resulting in a decomposition of the image into convex polygons. Here, we generate a polygonal decomposition of the height map. Intuitively, the detected line-segments, later included in the edges of the partition, correspond to regular height discontinuities in a given direction, i.e. to facades.

Using the probability map previously defined, we further assign a binary activation variable to each polygonal cell, indicating if it is part, or not, of a building footprint. Our strategy is similar to the model described in section 7.1.1. In order to simplify the partition, and decrease the number of cells, we finally apply a clustering algorithm that merges neighbor cells upon condition that there is no height discontinuity at their common border.

8.2.4 Building reconstruction

To obtain a LOD2 reconstruction from an oriented point cloud, from which we discard all points labelled as *vegetation*, we propose an algorithm in three steps.

First of all, we extract planar primitives from the point cloud. We apply the region-growing algorithm implemented in the CGAL library [OVJ+19]. A plane hypothesis is iteratively propagated from a point to its neighbors. It is accepted if it has a minimum number of inliers σ, with respect to a maximal point-to-plane distance ε and a normal deviation θ. If input points are noisy, more robust methods such as efficient RANSAC [SWK07] or structure-aware shape collapse [FLD18] can be considered. The threshold σ should be set

depending on the density of the cloud, so that σ points cover an area of 5 m^2 approximately, whereas ε and θ are typically set to 0.5 m and 25°. Once all planes have been extracted, we obtain a set of primitives represented by the planar convex hulls of the different sets of inliers associated to those planes.

The second step of our algorithm consists in computing a partition of the 3D space, defined by the previously extracted set of planar primitives. To this end, we rely on our kinetic approach described in chapter 6. However, for large datasets with millions of points, the primitive extraction step leads to the detection of thousands of primitives, and the simultaneous propagation of this huge set of primitives exceeds the capacities of our algorithm. Furthermore, vertical planes corresponding to facades cannot be extracted from the input point cloud, since the data is missing.

This is the reason why we split the spatial propagation problem into F subproblems, where F is the number of polygonal footprints returned by the building contouring procedure described in section 8.2.3. More precisely, for each footprint we get the list of primitives that intersect or are included in it by projection, and perform a spatial propagation restricted to the dimensions of the footprint. We obtain a set of F 3D subgraphs $G_1, G_2, \dots G_F$.

The third and final step of our pipeline consists in labelling the polyhedra of each subgraph G_i as inside or outside the buildings to reconstruct. The facets at the interface between outside and inside polyhedra then correspond to the output surface.

We use a voting scheme, based on the observation that in aerial datasets, points delimit the upper parts of the objects of interest. Let P_i be the set of polyhedrons of the subgraph G_i. For each polyhedron $p_j \in P_i$, where $j = 1, 2 \dots |P_i|$, we initialize a counter c_j to 0. All polyhedrons located below (resp. above) any plane inlier decrement (resp. increment) their counters.

Let us now consider a vector X with $|P_i|$ binary activation variables: $x_j = 1$ (resp. $x_j = 0$) if the polyhedron p_j is labelled as inside (resp. outside) a building to extract. We measure the quality of an output surface using a two-term energy of the form

$$U(X) = D(X) + \lambda V(X) \tag{8.1}$$

$D(X)$ is a data term that encourages the selection of a polyhedron p_j when $c_j < 0$, and its rejection when $c_j > 0$. We have:

$$D(X) = \frac{1}{|\mathcal{I}|} \sum_{j=1}^{|P_i|} d_j(x_j) \tag{8.2}$$

where

$$d_j(x_j) = \begin{cases} -c_j & \text{when } x_j = 0 \\ c_j & \text{when } x_j = 1 \end{cases} \tag{8.3}$$

and $|\mathcal{I}|$ is twice the number of inliers. $V(X)$, for its part, is a generalized Potts model that penalizes the total area of the surface:

$$V(X) = \frac{1}{A} \sum_{j \sim k} a_{jk} \cdot 1_{x_j \neq x_k} \tag{8.4}$$

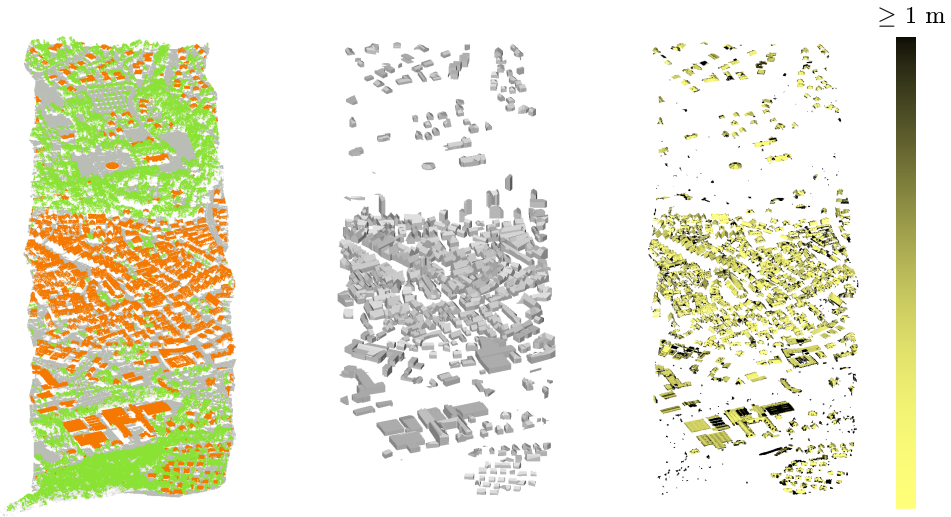
where $j \sim k$ denotes an adjacency relationship between two polyhedra p_j and p_k, a_{jk} is the surface of their common facet, and A is a normalization term defined as the sum of the areas of all facets of the subgraph G_i.

Given a balancing term $\lambda \in [0, 1]$, the optimal surface that minimizes energy U is determined by a min-cut algorithm [BK04]. A low value of λ returns a too large and too complex surface, while a high value of λ tends to shrink it. In our experiments, we typically set λ to 0.5.

8.3 Experiments

Datasets. We tested our algorithm on four datasets, representing various urban landscapes. The covered cities are listed in Table 8.1. The size is given in millions of points.

Qualitative results. We present in Figures 8.2, 8.3 and 8.4 the models generated by our algorithm for these cities. From a qualitative point of view, we obtain persuasive LOD2 reconstructions of most buildings. The Biberach and Vaihingen data-sets contain a lot of gable and hip roofs which are correctly approximated by our algorithm. As for the Portland and San Diego datasets, our technique also succeeds in determining intermediate levels in complex structures, as well as tilted roofs if any. Facades, which are almost completely missing in the input scans, are generally recovered by a single

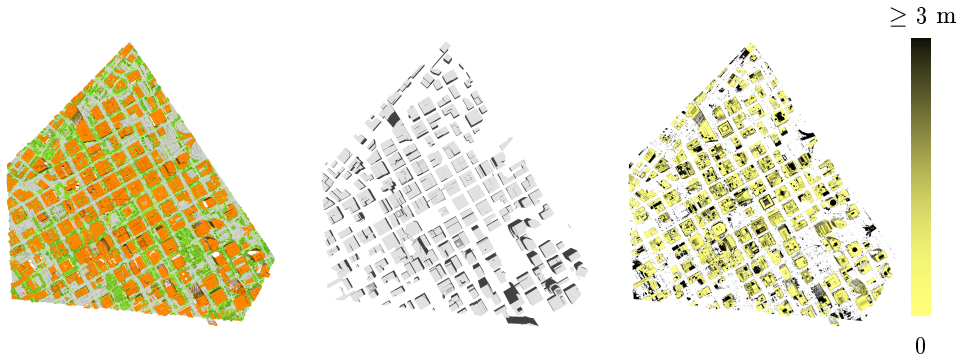


(a) Biberach.

(b) Vaihingen.

Figure 8.2: Results on European-style urban landscapes. Left column: classified point clouds. Points labelled as ground, vegetation or buildings are colored in gray, green and red, respectively. Center column: reconstructed models. Right column: altimetric error map, in which a darker color represents a larger error. From left to right: classified point cloud, reconstructed model, and altimetric error map.

plane in a given direction. Some of them, however, might be affected by an unwanted zig-zagging effect, reflecting errors in the building contouring



(a) Portland.

(b) San Diego.

Figure 8.3: Results on American-style urban landscapes.

City	Type	Size	Density (pts/m^2)
Biberach	Historical center	2.3M	3.0
Vaihingen	Residental area	7.3M	6.3
Portland	Downtown	8.7M	7.8
San Diego	Downtown	4.5M	1.6

Table 8.1: Presentation of the dataset.

procedure. Note that trees could be also reconstructed in 3D by template matching [VL12] to better represent the urban landscapes.

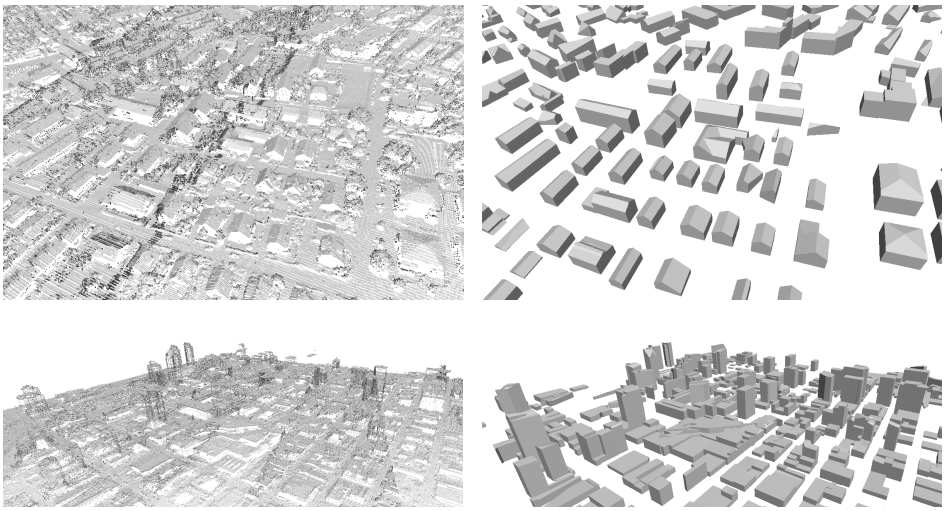


Figure 8.4: Close-ups on the reconstructed models. Top: residential area (Vaihingen). Bottom: urban landscape (San Diego).

Quantitative results. In Figures 8.2 and 8.3, we provide altimetric error maps for each dataset. To generate them, we restrict the input point cloud to elements labelled as rooftops, and compute the one-sided Hausdorff distance from every point to the reconstructed surface. This way, we can evaluate the precision of our method. We compute a *raw* reconstruction error: the mean one-sided Hausdorff distance. However, this measure may be biased. Indeed, isolated mislabelled points in the classification process, and a few buildings missed by the contouring algorithm and further ignored in the reconstruction phase, tend to overestimate the average reconstruction error. That is why we suggest, on an indicative basis, a *corrected* reconstruction error, which discards points located at more than 3 meters from a building which corresponds to the average height of a floor. Our measures are listed in Table 8.2.

The geometric error is typically caused by: (i) undetected superstructures on rooftops, (ii) uncorrectly approximated primitives in the plane extraction procedure (e.g. a unique plane approximates the two sides of a gable roof) and (iii) the resolution of the generated height map, that may shift the extracted footprints towards one direction or another.

City	Raw error (m)	Corrected error (m)
Biberach	0.85	0.31
Vaihingen	1.91	0.43
Portland	1.75	0.53
San Diego	2.34	0.62

Table 8.2: Geometric error for each dataset.

Performances. We list in Table 8.3 the performances of our algorithm for our two biggest datasets, Vaihingen and Portland, in terms of memory peak and running times. Measures were performed on a machine equiped with an Intel® Core™ i7-6700HQ processor clocked at 2.60 GHz and a 32 GB RAM. The obtained values demonstrate the ability of our algorithm to process large volumes of data in a short time.

	Vaihingen	Portland
Building contouring (s)	13.5	20.9
Building reconstruction (s)	509.3	872.8
Memory peak (GB)	2.2	2.6

Table 8.3: Performance measures for datasets Vaihingen and Portland.

Limitations. Despite its advantages, our algorithm suffers from a few shortcomings. Any error caused by an intermediate step of the pipeline has an impact on the final result. Parts of the cloud representing buildings mislabelled by the classifier will not be reconstructed. Besides, the reconstruction is very sensitive to building contouring errors: if some footprints or intermediate heights inside multi-level structures are missed in the coutouring process, then buildings will also be missing or badly approximated in the final result.

Besides, our primitive detection scheme only extracts planes from the input point cloud. This feature is sufficient for reconstructing accurately most buildings, but free-form shapes like domes or curved walls will only be approximated as a set of planar shapes. Small structures like dormers and

chimneys may also be ignored in the reconstructed model, if the minimal number of inliers by the plane extraction procedure is too high.

8.4 Conclusions

We presented a pipeline for automatically reconstructing a urban scene from an airborne Lidar scan at the level of detail LOD2. We used the kinetic approach described in previous chapters, in which a set of predetected line-segments and planar polygons progressively extend to decompose the 2D and 3D spaces into cells that are labelled and assembled in our final model. Our approach is fast, scalable and delivers simple polyhedral meshes. It returns promising results on various datasets, representing different types of urban environments.

In future works, we plan to refine the building contouring algorithm in order to simplify and improve the accuracy of our reconstructed models. We might resort to deep-learning-based methods to this end. Another research path would consist in integrating non-linear primitives to our kinetic scheme to achieve better reconstructions of free-form structures.

Chapter 9

Conclusion and perspectives

9.1 Summary

This thesis addressed the problem of geometric modeling of urban objects from physical measurements, and their represention in an accurate and compact manner. We presented two pipelines, one for the vectorization of regions of interest in images, another for the generation of concise polygonal meshes from point clouds.

Provided a set of predetected shapes, the core contribution of our work lies in the design of kinetic data structures for decomposing the 2D and 3D bounding volumes into partitions of polygonal cells. Experiments on images demonstrate that our algorithm offers an interesting alternative to superpixel decomposition techniques. Indeed, the latter typically operate at pixel level and produce homogeneously-sized cells with complex boundaries [LTRC11, ASS+12]. By reasoning at the scale of geometric shapes, our method returns lighter and more meaningful partitions. Capable of processing large satellite images with several dozen millions pixels in a few minutes only, our algorithm is scalable and computationally efficient.

Another range of experiments emphasizes the benefits of kinetic data structures for 3D space partitioning. Our algorithm solves the scalability issue induced by a naive decomposition of a bounding volume using infinite planes [CLP10, NW17]. It produces a partition with a small number of polyhedral cells compared with an exhaustive decomposition, or alternative strategies constructing for instance a Delaunay triangulation of an input point cloud [LA13, VKVLV13]. As a shape assembling algorithm, our approach forms in some way a bond between slicing-based methods and connectivity-based methods that analyze an adjacency graph in a set of input planar shapes, but that are prone to linkage errors. Note that our kinetic frameworks return 2D and 3D partitions whose cells are convex by

construction. Convexity can be a valuable property for specific applications, including for instance physical simulations.

Secondary contributions of our work include a global regularization procedure for optimizing geometric relationships in a set of predetected shapes, and a cell selection model for extracting objects from the kinetic partions based on graph cuts. In 2D, we use a saliency map to vectorize regions of interest in images. Although very simple, our model can correctly vectorize organic shapes as well as man-made structures, and provide a brief geometric description of such objects using a small number of edges. In 3D, we use a visibility criterion based on normal orientation in point clouds. This model offers a stable and efficient alternative to more complex energetic formulations, for instance based on integer programming [NW17]. As a result, we obtain a scalable pipeline that successfully approximates various freeform or structured objects as concise polygonal meshes. From qualitative and quantitative points of view, our pipeline can compete with existing polygonal surface reconstruction methods, but also with traditional surface approximation pipelines.

We underline the fact that, except for the 3D global regularization procedure, our algorithms are fully generic and flexible. We could show, for instance, the applicative potential of kinetic data structures in city modeling. By combining the object vectorization and polygonal mesh generation procedures, we were able to turn large-scale airborne Lidar scans into compact CityGML-LOD2 representations of urban scenes, while guaranteeing a relatively low altimetric error, at the scale of the meter. These promising results might open the door to practical uses for the generated models, such as environmental simulations or entertainment applications.

Our work suffers from some weaknesses. In particular, the object vectorization and mesh generation algorithms are mostly parameter-free methods, but strongly depend on the quality of the set of input shapes. If there exists no shape roughly approximating some part of an object of interest, kinetic data structures will create cells with heterogeneous contents, and therefore lead to a bad approximation of the objects. However, retrieving line-segments or planar shapes in some input data can be challenging when the latter is too noisy. We also observe that the cell selection techniques for

object extraction from kinetic partitions favor data fidelity and compacity of the output models, with respect to the total edge length or surface area, but omit the notion of regularity that could benefit to some applications.

Let us now present some perspectives opened by our work.

9.2 Perspectives

Generalization to non-linear shapes. A natural extension of this work would be to support more complex primitives in our kinetic data structures, including non-linear shapes.

The use of Bézier curves, splines, NURBS or other parametric functions would bring more versatility to the partitions, and would allow to approximate or reconstruct freeform objects with a better complexity-distortion tradeoff. Obviously, the efficient computation of collisions involving non-linear shapes represents the main challenge to overcome.

Learning primitives. As mentioned before, one of the main limitations of our vectorization and reconstruction pipelines is the fact that the quality of the approximation directly depends on the correctness of the predected set of shapes.

Indeed, if a line-segment or a planar polygon slightly drifts from its expected orientation, then the final result can be impacted. Our regularization procedure, though, can correct some faults observed in the initial arrangement of line-segments or planes, and enhance the visual quality of the result.

However, the main difficulty encountered by our algorithm is the absence of primitive in a certain location of the image or the point cloud. This problem is all the more critical for images that objects of interest must be roughly approximated by a minimal set of line-segments covering at least a part of each contour before starting the partition, in order to guarantee a good approximation. If it is not the case, then regions of interest will be badly approximated or ignored by the object contouring model.

Yet, obtaining this minimal set of line-segments might be a difficult task. Extracting relevant line-segments from noisy images and gradient maps is a difficult problem that requires thresholding operations. The same concern applies to 3D point clouds. The region growing implementation used in our experiments requires three parameters (minimal number of inliers,

maximal point-to-plane distance and normal deviation) that are, for now, manually set. How to generalize the choice of such detection parameters ? We believe that methods learning the characteristics of primitives, either in images [HWZ+18, XBW+19] or point clouds [FLD18, LSD+19] offer an interesting research path that should be further investigated.

Cell refinement. An alternative strategy for improving the quality of the partition despite the misdetection or absence of primitives leads to the refinement of the cells, using split and merge operations.

In 2D, analyzing the colorimetric distribution within a polygonal cell can constitute a good hint for detecting heterogeneous contents. We may define a split operator on this basis. The latter can integrate geometric reasoning: for instance, a cell is more likely to be divided if it is located near two collinear line-segments, with one on each side of the cell. Likewise, we can define a merge operator for merging two cells that represent the same object.

Such a mechanism would reduce for sure undersegmentation errors, but also improve the quality of the object contouring results.

Geometry reinforcement. Another idea, which has not been explored within the frame of this thesis, would consist in improving the quality of the geometric models produced by the vectorization and 3D reconstruction pipelines by reinforcing geometric regularities.

We have a global regularization procedure at our disposal, which optimizes parallelism, orthogonality and collinearity relationships among a set of detected line-segments in an image. However, the object contouring model described in section 7.1 doesn't exploit the geometric relationships between the cells and edges of the partition. Penalizing irregularities will require to adapt our energy formulation and will lead to a more complex formulation, but many applications would benefit from the extraction of regularized contours. In particular, the urban modeling pipeline described in chapter 8 will return more compact and accurate models with regularized building footprints.

The same criticism can be formulated about the surface extraction model in section 7.2, since our formulation doesn't encourage the construction of regular surfaces, coming up with geometric guarantees. We plan to address these problems in some near future.

Applications of kinetic data structures. Finally, we plan to investigate on further applications of kinetic data structures.

We may cite, for instance, mesh repairing. Indeed, CAD models can be hampered by geometric insanities like holes, edge or vertex duplication, self-intersecting or redundant facets [BS96, Att10]. Yet, the construction of a kinetic partition from a flawed mesh, followed by the subsequent reconstruction step, may lead to the creation of a repaired model. Inspired by the work of Häne *et al* [HZC+13], another research path might include joint 3D reconstruction and semantic classification with N classes from our kinetic partitions, using a more advanced optimization framework.

Publications

This thesis is supported by the following publications:

- Jean-Philippe Bauchet, Florent Lafarge. KIPPI: KInetic Polygonal Partitioning of Images. In *Proceedings of Computer Vision and Pattern Recognition*, 2018.
- Jean-Philippe Bauchet, Florent Lafarge. City Reconstruction from Airborne LiDAR: A Computational Geometry Approach. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2019.
- Jean-Philippe Bauchet, Florent Lafarge. Kinetic Shape Reconstruction. Submitted to *Transactions on Graphics*, 2019.

Appendix A

Appendix

This appendix provides a detailed pseudo-code illustrating the different steps of the kinetic partitioning algorithm presented in chapter 6.

A.1 Polyhedral partitioning in 3D: a pseudo-code

Notations. Let:

- $\boldsymbol{P} = (P_1, .., P_N)$, be the input set of N convex polygons;
- $\overline{\boldsymbol{P}} = (\overline{P}_1, .., \overline{P}_n)$, be the set of n (infinite) supporting planes of polygons $\boldsymbol{P}$ with $n \leq N$;
- $\mathcal{P} = (\mathcal{P}_1, .., \mathcal{P}_n)$, be the set of 2D polygonal partitions with $\mathcal{P}_i$ defined in the supporting plane $\overline{P_i}$ by the triplet (L_i, T_i, S_i) :
  - L_i is the set of intersection lines $L_{ij} = \overline{P_i} \cap \overline{P_j}$;
  - T_i is the set of intersection-free polygons that propagate on the plane $\overline{P_i}$;
  - S_i is the set of line-segments $s_{ij} = \overline{P_i} \cap T_j$;
- Q, be the a global priority queue;
- $\mathcal{C}$, be the output partition of polyhedra;
- $\mathcal{F}$, be the set of polygonal facets contained in $\mathcal{C}$.

Some comments on the pseudo-code. An illustration of the initialization procedure can be found in Figure 6.2 of the paper. On the left side, the blue polygon is an input convex polygon P_i in the plane $\overline{P_i}$. On the right side, the dashed lines, the line-segments and the four blue intersection-free polygons are elements of L_i, S_i and T_i, respectively. Among the collition cases illustrated in Figure 6.3 of the paper, note that the initial polygon decomposition avoids considering collision case (d) in practice.

Algorithm 3 Kinetic partitiong (part 1/2)

1: $\boldsymbol{P} \leftarrow (P_1, P_2, \ldots, P_n)$
2: $\overline{\boldsymbol{P}} \leftarrow (\overline{P_1}, \overline{P_2}, \ldots, \overline{P_n})$
3: $\mathcal{P} \leftarrow \emptyset$
4: $Q \leftarrow \emptyset$
5:
6: **function** INITIALIZATION
7: **for** $\overline{P_i} \in \overline{\boldsymbol{P}}$ **do**
8: $L_i \leftarrow \bigcup_{j \neq i} L_{ij}$ where $L_{ij} = \overline{P_i} \cap \overline{P_j}$
9: $T_i \leftarrow \{P_i\}$
10: $S_i \leftarrow \emptyset$
11: $\mathcal{P}_i \leftarrow (L_i, T_i, S_i)$
12: Add $\mathcal{P}_i$ to $\mathcal{P}$
13: **end for**
14: **for** $\mathcal{P}_i \in \mathcal{P}$ **do**
15: **for** $L_{ij} \in L_i$ **do**
16: **for** $P \in T_i$ **do**
17: **if** $L_{ij} \cap P \neq \emptyset$ **then**
18: Create sliding or frozen vertices along L_{ij}
19: Split P into subpolygons P_1, P_2
20: Add P_1, P_2 to T_i
21: Remove P from T_i
22: Add line-segment $s = P_1 \cap P_2$ to S_j
23: **end if**
24: **end for**
25: **end for**
26: **for** $P \in T_i$ **do**
27: **for** each non-frozen vertex $v \in P$ **do**
28: Determine the 3 next events $e_{v,j} = \{t \mid v(t) \cap L_{ij} \neq \emptyset\}$ where $L_{ij} \in L_i$
29: Add these events to Q
30: **end for**
31: **end for**
32: **end for**
33: **end function**

Algorithm 4 Kinetic partitioning (part 2/2)

function PROCESSING_EVENTS
  while $Q \neq \emptyset$ **do**
    Pop the vertex v intersecting the line L_{ij} from Q
    Get the polygon P of T_i containing v
    Determine the collision case (see Figure 4)
    Update P with sliding and/or frozen vertices
    Determine the 3 next events of the new vertices of P
    Add these events to Q
    Add line-segment $s = \overline{P}_j \cap P$ to S_j
    if CROSSING_CONDITION(P, L_{ij}) = TRUE **then**
      Add a new polygon P' to T_i
      Determine the 3 next events of the vertices of P'
      Add these events to Q
      Add line-segment $s' = \overline{P}_j \cap P'$ to S_j
    end if
    if v is not frozen **then**
      if Q has no more events involving v **then**
        Determine the 3 next events involving v
        Add these events to Q
      end if
    else
      Remove all future events of v from Q
    end if
  end while
end function

function FINALIZATION
  $\mathcal{F} \leftarrow \emptyset$
  $\mathcal{C} \leftarrow \emptyset$
  for $\mathcal{P}_i \in \mathcal{P}$ **do**
    Assemble adjacent polygons of T_i into facets $\mathcal{F}_i$
    Add $\mathcal{F}_i$ to $\mathcal{F}$
  end for
  Assemble adjacent facets of $\mathcal{F}$ into polyhedra C
end function

Bibliography

[AFS+11] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. Seitz, and R. Szeliski. Building rome in a day. *Communications of the ACM*, 54(10), 2011. (Cited on page 6.)

[AGG+10] P. Agarwal, J. Gao, L. Guibas, H. Kaplan, V. Koltun, N. Rubin, and M. Sharir. Kinetic stable delaunay graphs. In *Proc. of Symposium on Computational Geometry (SoCG)*, 2010. (Cited on page 46.)

[ALKF18] D. Acuna, H. Ling, A. Kar, and S. Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on page 12.)

[AS17] R. Achanta and S. Susstrunk. Superpixels and polygons using simple non-iterative clustering. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. (Cited on pages 13, 14, 50, 51 and 52.)

[ASF+13] M. Arikan, M. Schwarzler, S. Flory, M. Wimmer, and S. Maierhofer. O-snap: Optimization-based snapping for modeling architecture. *Trans. on Graphics*, 32(1), 2013. (Cited on page 21.)

[ASS+12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 34(11), 2012. (Cited on pages 13 and 119.)

[ATQE17] E. Almazan, R. Tal, Y. Qian, and J. Elder. Mcmlsd: A dynamic programming approach to line segment detection. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, pages 2031–2039, 2017. (Cited on page 15.)

[Att10] M. Attene. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer*, 26(11):1393–1406, 2010. (Cited on page 123.)

[Bal81] D. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981. (Cited on page 20.)

[BBPDM08] M. Brédif, D. Boldo, M. Pierrot-Deseilligny, and H. Maître. 3d building model fitting using a new kinetic framework. *arXiv:0805.0648*, 2008. (Cited on page 46.)

[BDLGM14] A. Boulch, M. De La Gorce, and R. Marlet. Piecewise-planar 3d reconstruction with edge and corner regularization. *Computer Graphics Forum*, 33(5), 2014. (Cited on pages 21, 22 and 77.)

[BGH99] J. Basch, L. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1), 1999. (Cited on page 45.)

[BK04] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, (9), 2004. (Cited on pages 73, 77 and 113.)

[BM92] P. Besl and N. McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, 1992. (Cited on page 5.)

[BS96] G. Butlin and C. Stops. Cad data repair. In *Proc. of the International Meshing Roundtable*, 1996. (Cited on page 123.)

[BSL+15] F. Biljecki, J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin. Applications of 3D city models: state of the art review. *ISPRS International Journal of Geo-Information*, 4(4), 2015. (Cited on page 107.)

[BSRVG14] A. Bodis-Szomoru, H. Riemenschneider, and L. Van Gool. Fast, approximate piecewise-planar modeling based on sparse structure-from-motion and superpixels. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2014. (Cited on page 50.)

[BSRVG15] A. Bodis-Szomoru, H. Riemenschneider, and L. Van Gool. Superpixel meshes for fast edge-preserving surface reconstruction. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on page 18.)

[BTS+17] M. Berger, A. Tagliasacchi, L. Seversky, P. Alliez, G. Guennebaud, J. Levine, A. Sharf, and C. Silva. A survey of surface reconstruction from point clouds. *Computer Graphics Forum*, 36(1), 2017. (Cited on page 18.)

[Can87] J. Canny. A computational approach to edge detection. In *Readings in computer vision*. Elsevier, 1987. (Cited on page 15.)

[CC08] J. Chen and B. Chen. Architectural modeling from sparsely scanned range data. *International Journal of Computer Vision (IJCV)*, 78(2-3), 2008. (Cited on page 21.)

[CKUF17] L. Castrejon, K. Kundu, R. Urtasun, and S. Fidler. Annotating object instances with a polygon-rnn. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2017. (Cited on pages 12 and 73.)

[CLP10] A.-L. Chauve, P. Labatut, and J.-P. Pons. Robust piecewise-planar 3D reconstruction and completion from large-scale unstructured point data. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2010. (Cited on pages 21, 22, 59, 68, 77, 90, 91, 92 and 119.)

[CMH+14] M. Cheng, N. Mitra, X. Huang, P. Torr, and S. Hu. Global contrast based salient region detection. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 37(3):569–582, 2014. (Cited on page 13.)

[Cra07] A. Cracknell. *Introduction to remote sensing*. CRC press, 2007. (Cited on page 4.)

[CSAD04] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *Proc. of SIGGRAPH*, 2004. (Cited on pages 18, 93 and 94.)

[CSALM13] D. Chen, P. Sitthi-Amorn, J. Lan, and W. Matusik. Computing and fabricating multiplanar models. In *Computer Graphics Forum*, volume 32, 2013. (Cited on page 18.)

[CY00] J. Coughlan and A. Yuille. The manhattan world assumption: Regularities in scene statistics which enable bayesian inference. In *Proc. of Neural Information Processing Systems (NIPS)*, 2000. (Cited on page 18.)

[CYL17] N.-G. Cho, A. Yuille, and S.-W. Lee. A novel linelet-based representation for line segment detection. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 40(5), 2017. (Cited on page 15.)

[DAPP17] S. Duchene, C. Aliaga, T. Pouli, and P. Perez. Mixed illumination analysis in single image for interactive color grading. In *Proc. of Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, 2017. (Cited on page 45.)

[DDG14] A. Diakité, G. Damiand, and G. Gesquière. Automatic Semantic Labelling of 3D Buildings Based on Geometric and Topological Information. In *Proc. of 9th International 3DGeoInfo Conference (3DGeoInfo)*, 2014. (Cited on page 23.)

[DDVM14] A. Diakité, G. Damiand, and D. Van Maercke. Topological reconstruction of complex 3d buildings and automatic extraction of levels of detail. In *Eurographics Workshop on Urban Data Modelling and Visualisation*, pages 25–30, 2014. (Cited on page 22.)

[dGCSAD11] F. de Goes, D. Cohen-Steiner, P. Alliez, and M. Desbrun. An optimal transport approach to robust reconstruction and simplification of 2d shapes. *Computer Graphics Forum*, 30(5), 2011. (Cited on page 13.)

[DH71] R. Duda and P. Hart. Use of the hough transformation to detect lines and curves in pictures. Technical report, Sri International Menlo Park Ca Artificial Intelligence Center, 1971. (Cited on pages 15 and 20.)

[DHH11] M. Dubska, A. Herout, and J. Havel. Pclines – line detection using parallel coordinates. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2011. (Cited on page 15.)

[DL14] G. Damiand and P. Lienhardt. *Combinatorial maps: efficient data structures for computer graphics and image processing.* CRC Press, 2014. (Cited on page 22.)

[DL15] L. Duan and F. Lafarge. Image partitioning into convex polygons. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on pages 14, 16, 28, 50, 51, 52, 75 and 76.)

[DL16] L. Duan and F. Lafarge. Towards large-scale city reconstruction from satellites. In *Proc. of European Conference on Computer Vision (ECCV)*, 2016. (Cited on pages 45 and 108.)

[DMM07] A. Desolneux, L. Moisan, and J.-M. Morel. *From gestalt theory to image analysis: a probabilistic approach*, volume 34. Springer Science & Business Media, 2007. (Cited on page 15.)

[DP73] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitizedline or its caricature. *Cartographica: The International Journal for Geographic Information*, 10(2), 1973. (Cited on pages 13, 75 and 76.)

[EE99] D. Eppstein and J. Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete and Computational Geometry*, 22(4), 1999. (Cited on page 48.)

[FH+15] Y. Furukawa, C. Hernández, et al. Multi-view stereo: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 9(1-2), 2015. (Cited on page 6.)

[FK17] J. Forsythe and V. Kurlin. Convex constrained meshes for superpixel segmentations of images. *Journal of Electronic Imaging*, 26(6), 2017. (Cited on page 16.)

[FKF16] J. Forsythe, V. Kurlin, and A. Fitzgibbon. Resolution-independent superpixels based on convex constrained meshes without small angles. In *Proc. of International Symposium on Visual Computing (ISVC)*, 2016. (Cited on page 16.)

[FLBA19] J.-D. Favreau, F. Lafarge, A. Bousseau, and A. Auvolat. Extracting geometric structures in images with delaunay point processes. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2019. (Cited on page 17.)

[FLD18] H. Fang, F. Lafarge, and M. Desbrun. Planar Shape Detection at Structural Scales. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on pages 20, 111 and 122.)

[FP09] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 32(8), 2009. (Cited on page 6.)

[GA12] R. Guerreiro and P. Aguiar. Connectivity-enforcing hough transform for the robust extraction of line segments. *Trans. on Image Processing*, 21(12), 2012. (Cited on page 16.)

[GH97] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proc. of SIGGRAPH*, 1997. (Cited on pages 18, 80, 93 and 94.)

[GKNH12] G. Gröger, T. Kolbe, C. Nagel, and K.-H. Häfele. Ogc city geography markup language (citygml) encoding standard. 2012. (Cited on page 107.)

[GL19] S. Giraudot and F. Lafarge. Classification. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14 edition, 2019. (Cited on page 110.)

[GO19] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019. (Cited on page 89.)

[GP12] G. Groger and L. Plumer. Citygml – interoperable semantic 3d city models. *Journal of Photogrammetry and Remote Sensing*, 71, 2012. (Cited on page 29.)

[GS97] T. Gevers and A. W. M. Smeulders. Combining region splitting and edge detection through guided delaunay image subdivision. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 1997. (Cited on page 16.)

[GT18] N. Girard and Y. Tarabalka. End-to-End Learning of Polygons for Remote Sensing Image Classification. In *Proc. of IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2018. (Cited on page 12.)

[Gui04] L. Guibas. Kinetic data structures. In *Handbook of Data Structures and Applications*, 2004. (Cited on pages 45 and 59.)

[GW03] M. Gertz and S. Wright. Object-oriented software for quadratic programming. *Trans. on Mathematical Software*, 29(1), 2003. (Cited on pages 33 and 43.)

[HB03] L. Hermes and J. Buhmann. A minimum entropy approach to adaptive image polygonization. *Trans. on Image Processing*, 12(10), 2003. (Cited on page 17.)

[HBS13] H. Huang, C. Brenner, and M. Sester. A generative statistical approach to automatic 3d building roof reconstruction from laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 79, 2013. (Cited on page 109.)

[HGSP13] A. Henn, G. Gröger, V. Stroh, and L. Plümer. Model driven reconstruction of roofs from sparse lidar point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76, 2013. (Cited on page 109.)

[HHRB11] D. Holz, S. Holzer, R. Rusu, and S. Behnke. Real-time plane segmentation using rgb-d cameras. In *Robot Soccer World Cup*, pages 306–317. Springer, 2011. (Cited on page 20.)

[HJS+14] J. Huang, T. Jiang, Z. Shi, Y. Tong, H. Bao, and M. Desbrun. l_1-based construction of polycube maps from complex shapes. *Trans. on Graphics*, 33(3), 2014. (Cited on pages 18 and 19.)

[HK06] A. Hornung and L. Kobbelt. Robust reconstruction of watertight 3 d models from non-uniformly sampled point clouds

without normal information. In *Proc. of Symposium on Geometry Processing (SGP)*, pages 41–50. Citeseer, 2006. (Cited on page 22.)

[HK10] N. Haala and M. Kada. An update on automatic 3d building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(6), 2010. (Cited on page 107.)

[HKLP09] V. Hiep, R. Keriven, P. Labatut, and J.-P. Pons. Towards high-resolution large-scale multi-view stereo. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2009. (Cited on page 6.)

[HSDF15] J. Heinly, J. Schonberger, E. Dunn, and J.-M. Frahm. Reconstructing the world* in six days*(as captured by the yahoo 100 million image dataset). In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on page 6.)

[HSMS13] R. Hulik, M. Spanel, Z. Materna, and P. Smrz. Continuous plane detection in point-cloud data based on 3d hough transform. *Journal of Visual Communication and Image Representation*, 25(1), 2013. (Cited on page 20.)

[HWZ+18] K. Huang, Y. Wang, Z. Zhou, T. Ding, S. Gao, and Y. Ma. Learning to parse wireframes in images of man-made environments. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on pages 16 and 122.)

[HZC+13] C. Hane, C. Zach, A. Cohen, R. Angst, and M. Pollefeys. Joint 3d scene reconstruction and class segmentation. In *Proc of. Computer Vision and Pattern Recognition (CVPR)*, 2013. (Cited on page 123.)

[IKH+11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proc. of Symposium on User interface software and technology*, pages 559–568, 2011. (Cited on page 7.)

[IYF15] S. Ikehata, H. Yang, and Y. Furukawa. Structured indoor modeling. In *Proc. of International Conference on Computer Vision (ICCV)*, 2015. (Cited on page 18.)

[JTV+15] C. Jiang, C. Tang, A. Vaxman, P. Wonka, and H. Pottmann. Polyhedral patterns. *Trans. on Graphics*, 34(6), 2015. (Cited on page 18.)

[JWWP14] C. Jiang, J. Wang, J. Wallner, and H. Pottmann. Freeform honeycomb structures. *Computer Graphics Forum*, 33, 2014. (Cited on page 18.)

[KBH06] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symposium on Geometry Processing*, 2006. (Cited on pages 6, 18, 22 and 80.)

[KFWM17] T. Kelly, J. Femiani, P. Wonka, and N. Mitra. Bigsur: Large-scale structured urban reconstruction. *Trans. on Graphics*, 36(6), 2017. (Cited on page 108.)

[KH13] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *Trans. on Graphics*, 32(3), 2013. (Cited on pages 18 and 94.)

[KM09] M. Kada and L. McKinley. 3d building reconstruction from lidar based on a cell decomposition approach. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(Part 3), 2009. (Cited on page 108.)

[KM19] V. Kurlin and G. Muszynski. A persistence-based approach to automatic detection of line segments in images. In *International Workshop on Computational Topology in Image Context*, 2019. (Cited on page 15.)

[KPZK17] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *Trans. on Graphics*, 36(4), 2017. (Cited on pages 6 and 80.)

[KvLS07] R. Kluszczyński, M. N. M. van Lieshout, and T. Schreiber. Image segmentation by polygonal markov fields. *Annals of the*

Institute of Statistical Mathematics, 59(3), 2007. (Cited on page 12.)

[KYZB18] A. Kaiser, J. Ybanez Zepeda, and T. Boubekeur. A survey of simple geometric primitives detection methods for captured 3d data. *Computer Graphics Forum*, 37, 2018. (Cited on page 19.)

[LA13] F. Lafarge and P. Alliez. Surface reconstruction through point set structuring. In *Computer Graphics Forum*, volume 32, 2013. (Cited on pages 21, 22 and 119.)

[LB07] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2007. (Cited on page 22.)

[LCC+10] T. Liu, M. Carlberg, G. Chen, J. Chen, J. Kua, and A. Zakhor. Indoor localization and visualization using a human-operated backpack system. In *2010 International Conference on Indoor Positioning and Indoor Navigation*, pages 1–10. IEEE, 2010. (Cited on page 5.)

[LDZPD08] F. Lafarge, X. Descombes, J. Zerubia, and M. Pierrot-Deseilligny. Automatic building extraction from dems using an object approach and application to the 3d-city modeling. *ISPRS Journal of photogrammetry and remote sensing*, 63(3), 2008. (Cited on page 12.)

[LDZPD10] F. Lafarge, X. Descombes, J. Zerubia, and M. Pierrot-Deseilligny. Structural approach for building reconstruction from a single DSM. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 32(1), 2010. (Cited on page 109.)

[LGK+19] H. Ling, J. Gao, A. Kar, W. Chen, and S. Fidler. Fast interactive object annotation with curve-gcn. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on page 12.)

[Lin00] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proc. of SIGGRAPH*, 2000. (Cited on page 18.)

[LM11] F. Lafarge and C. Mallet. Building large urban environments from unstructured point data. In *Proc. of the International Conference on Computer Vision (ICCV)*, Barcelona, Spain, 2011. (Cited on page 108.)

[LO15] F. Limberger and M. Oliveira. Real-time detection of planar regions in unorganized point clouds. *Pattern Recognition*, 48(6):2043–2053, 2015. (Cited on page 20.)

[Low04] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004. (Cited on page 6.)

[LPK09] P. Labatut, J.-P. Pons, and R. Keriven. Robust and Efficient Surface Reconstruction From Range Data. *Computer Graphics Forum*, 28(8), 2009. (Cited on page 77.)

[LQQ+18] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on page 13.)

[LSD10] A. Levinshtein, C. Sminchisescu, and S. Dickinson. Optimal contour closure by superpixel grouping. In *Proc. of European Conference on Computer Vision (ECCV)*, 2010. (Cited on pages 13 and 73.)

[LSD+19] L. Li, M. Sung, A. Dubrovina, L. Yi, and L. Guibas. Supervised Fitting of Geometric Primitives to 3D Point Clouds . In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on pages 20 and 122.)

[LTRC11] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa. Entropy rate superpixel segmentation. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2011. (Cited on pages 13, 14, 50, 52 and 119.)

[LWC+11] Y. Li, X. Wu, Y. Chrysathou, A. Sharf, D. Cohen-Or, and N. Mitra. Globfit: consistently fitting primitives by discovering global relations. In *Trans. on Graphics*, volume 30, 2011. (Cited on page 20.)

[LWL18] Z. Li, J. Wegner, and A. Lucchi. Polymapper: Extracting city maps using polygons. *arXiv preprint arXiv:1812.01497*, 2018. (Cited on pages 12 and 13.)

[LY16] G. Li and Y. Yu. Deep contrast learning for salient object detection. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on page 13.)

[MB98] E. Mortensen and W. Barrett. Interactive segmentation with intelligent scissors. *Graphical models and image processing*, 60(5), 1998. (Cited on page 12.)

[MF97] T. M. Murali and A. Funkhouser. Consistent solid and boundary representations from arbitrary polygonal data. In *Proc. of Symposium on Interactive 3D Graphics (SI3D)*, 1997. (Cited on page 21.)

[MFTM01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. of International Conference on Computer Vision (ICCV)*, 2001. (Cited on pages 50 and 51.)

[MLM01] D. Marshall, G. Lukacs, and R. Martin. Robust segmentation of primitives from range data in the presence of geometric degeneracy. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 23(3), 2001. (Cited on page 20.)

[MMBM15] A. Monszpart, N. Mellado, G. Brostow, and N. Mitra. Rapter: Rebuilding man-made scenes with regular arrangements of planes. *Trans. on Graphics*, 34(4), 2015. (Cited on page 20.)

[MMP16] C. Mura, O. Mattausch, and R. Pajarola. Piecewise-planar reconstruction of multi-room interiors with arbitrary wall arrangements. *Computer Graphics Forum*, 35(7), 2016. (Cited on page 21.)

[MMV+14] C. Mura, O. Mattausch, A. Villanueva, E. Gobbetti, and R. Pajarola. Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts. *Computers & Graphics*, 44, 2014. (Cited on page 45.)

[MWA+13] P. Musialski, P. Wonka, D. Aliaga, M. Wimmer, L. Van Gool, and W. Purgathofer. A survey of urban reconstruction. *Computer Graphics Forum*, 32(6), 2013. (Cited on pages 8, 18 and 107.)

[NIH+11] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, volume 11, 2011. (Cited on page 7.)

[NP12] P. Neubert and P. Protzel. Superpixel benchmark and comparison. In *Proc. Forum Bildverarbeitung*, 2012. (Cited on page 51.)

[NW17] L. Nan and P. Wonka. Polyfit: Polygonal surface reconstruction from point clouds. In *Proc. of International Conference on Computer Vision (ICCV)*, 2017. (Cited on pages 21, 22, 59, 68, 88, 89, 90, 91, 92, 97, 119 and 120.)

[OLA14] S. Oesau, F. Lafarge, and P. Alliez. Indoor scene reconstruction using feature sensitive primitive extraction and graph-cut. *ISPRS Journal of Photogrammetry and Remote Sensing*, 90:68–82, 2014. (Cited on page 21.)

[OLA16] S. Oesau, F. Lafarge, and P. Alliez. Planar shape detection and regularization in tandem. In *Computer Graphics Forum*, volume 35, 2016. (Cited on pages 16 and 20.)

[Ope19] OpenAerialMap. Openaerialmap - the open collection of aerial imagery. `https://openaerialmap.org/`, 2019 (accessed August 19, 2019). (Cited on pages 54, 56 and 57.)

[OVJ+19] S. Oesau, Y. Verdie, C. Jamin, P. Alliez, F. Lafarge, and S. Giraudot. Point set shape detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.14 edition, 2019. (Cited on pages 35 and 111.)

[PCSS14] T.-T. Pham, T.-J. Chin, K. Schindler, and D. Suter. Interacting geometric priors for robust multimodel fitting. *Trans. on Image Processing*, 23(10), 2014. (Cited on page 16.)

[Pou13] C. Poullis. A framework for automatic modeling from point cloud data. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 35(11), 2013. (Cited on page 108.)

[PS05] L. Prasad and A. Skourikhine. Vectorized image segmentation via trixel agglomeration. In *International Workshop on Graph-Based Representations in Pattern Recognition*, 2005. (Cited on page 17.)

[PY09] C. Poullis and S. You. Automatic reconstruction of cities from remote sensor data. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2009. (Cited on page 108.)

[QYSG17] C. Qi, L. Yi, H. Su, and L. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. (Cited on page 20.)

[QZH+19] X. Qin, Z. Zhang, C. Huang, C. Gao, M. Dehghan, and M. Jagersand. Basnet: Boundary-aware salient object detection. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on page 13.)

[RBDD18] S. Rodriguez, A. Bousseau, F. Durand, and G. Drettakis. Exploiting repetitions for image-based rendering of facades. In *Computer Graphics Forum*, volume 37, 2018. (Cited on page 45.)

[RKB04] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *Trans. on Graphics*, volume 23, 2004. (Cited on pages 12, 75 and 76.)

[RS13] Z. Ren and G. Shakhnarovich. Image segmentation by cascaded region agglomeration. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2013. (Cited on page 13.)

[RSG+14] F. Rottensteiner, G. Sohn, M. Gerke, J. Wegner, U. Breitkopf, and J. Jung. Results of the isprs benchmark on urban object detection and 3d building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93, 2014. (Cited on page 107.)

[RVDHV06] T. Rabbani, F. Van Den Heuvel, and G. Vosselman. Segmentation of point clouds using smoothness constraint. *ISPRS*, 36(5), 2006. (Cited on page 20.)

[SCD+06] S. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2006. (Cited on page 6.)

[SCF14] X. Sun, M. Christoudias, and P. Fua. Free-shape polygonal object localization. In *Proc. of European Conference on Computer Vision (ECCV)*, 2014. (Cited on pages 11 and 73.)

[SDK09] R. Schnabel, P. Degener, and R. Klein. Completion and reconstruction with primitive shapes. In *Computer Graphics Forum*, volume 28, 2009. (Cited on page 22.)

[SF16] J. Schönberger and J.-M. Frahm. Structure-from-Motion Revisited. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2016. (Cited on pages 6 and 80.)

[She02] J. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational geometry*, 22(1-3), 2002. (Cited on page 16.)

[SHT08] G. Sohn, X. Huang, and V. Tao. Using a binary space partitioning tree for reconstructing polyhedral building models from airborne lidar data. *Photogrammetric Engineering & Remote Sensing*, 74(11), 2008. (Cited on page 108.)

[SLA15] D. Salinas, F. Lafarge, and P. Alliez. Structure-Aware Mesh Decimation. *Computer Graphics Forum*, 34(6), 2015. (Cited on pages 17, 18, 93, 94 and 108.)

[SMP07] S. Sinha, P. Mordohai, and M. Pollefeys. Multi-view stereo via graph cuts on the dual of an adaptive tetrahedral mesh. In *Proc. of International Conference on Computer Vision (ICCV)*, 2007. (Cited on page 6.)

[SSS06] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *Trans. on Graphics*, volume 25, 2006. (Cited on page 6.)

[ST10] D. San and M. Turker. Building extraction from high resolution satellite images using hough transform. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science*, 38(Part 8), 2010. (Cited on page 13.)

[SWF11] F. Schindler, W. Worstner, and J.-. Frahm. Classification and reconstruction of surfaces from point clouds of man-made objects. In *Proc. of International Conference on Computer Vision (ICCV) Workshops*, 2011. (Cited on page 21.)

[SWK07] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. In *Computer Graphics Forum*, volume 26, 2007. (Cited on pages 19 and 111.)

[SWWK08] R. Schnabel, R. Wessel, R. Wahl, and R. Klein. Shape recognition in 3d point-clouds. 2008. (Cited on page 19.)

[SZFP16] J. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *Proc. of European Conference on Computer Vision (ECCV)*, 2016. (Cited on page 6.)

[THCM04] M. Tarini, K. Hormann, P. Cignoni, and C. Montani. Polycube-maps. *Trans. on Graphics*, 23(3), 2004. (Cited on page 18.)

[TLJ+18] W.-C. Tu, M.-Y. Liu, V. Jampani, D. Sun, S.-Y. Chien, M.-H. Yang, and J. Kautz. Learning superpixels with segmentation-aware affinity loss. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2018. (Cited on page 13.)

[TMAT18] O. Tasar, E. Maggiori, P. Alliez, and Y. Tarabalka. Polygonization of binary classification maps using mesh approximation with right angle regularity. In *Proc. of IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2018. (Cited on page 13.)

[VdBBR+12] M. Van den Bergh, X. Boix, G. Roig, B. De Capitani, and L. Van Gool. SEEDS: Superpixels extracted via energy-driven sampling. In *Proc. of European Conference on Computer Vision (ECCV)*, 2012. (Cited on page 13.)

[vGJMR10] R. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 32(4), 2010. (Cited on pages 15, 16, 31 and 34.)

[VKH06] V. Verma, R. Kumar, and S. Hsu. 3d building detection and modeling from aerial lidar data. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2006. (Cited on page 109.)

[VKVLV11] M. Van Kreveld, T. Van Lankveld, and R. Veltkamp. On the shape of a set of points and lines in the plane. *Computer Graphics Forum*, 30, 2011. (Cited on page 21.)

[VKVLV13] M. Van Kreveld, T. Van Lankveld, and R. Veltkamp. Watertight scenes from urban lidar and planar surfaces. In *Proc. of tSymposium on Geometry Processing (SGP)*. Eurographics Association, 2013. (Cited on pages 22 and 119.)

[VL12] Y. Verdie and F. Lafarge. Efficient Monte Carlo sampler for detecting parametric objects in large scenes. In *Proc. of the European Conference on Computer Vision (ECCV)*, Firenze, Italy, 2012. (Cited on page 115.)

[VLA15] Y. Verdie, F. Lafarge, and P. Alliez. LOD Generation for Urban Scenes. *Trans. on Graphics*, 34(3), 2015. (Cited on pages 22, 77 and 108.)

[WKSW05] S. Wang, T. Kubota, J. Siskind, and J. Wang. Salient closed boundary extraction with ratio contour. *Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 27(4), 2005. (Cited on page 12.)

[WM03] S. Wu and M. Marquez. A non-self-intersection Douglas-Peucker algorithm. In *Proc. of Symposium on Computer Graphics and Image Processing*, 2003. (Cited on page 13.)

[WPC18] R. Wang, J. Peethambaran, and D. Chen. Lidar point clouds to 3-d urban models : a review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(2), 2018. (Cited on page 108.)

[WWL+16] L. Wang, L. Wang, H. Lu, P. Zhang, and X. Ruan. Saliency detection with recurrent fully convolutional networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Proc. of European Conference on Computer Vision (ECCV)*, 2016. (Cited on page 13.)

[XBW+19] N. Xue, S. Bai, F. Wang, G.-S. Xia, T. Wu, and L. Zhang. Learning attraction field representation for robust line segment detection. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on pages 16 and 122.)

[XEV14] B. Xiong, S. Elberink, and G. Vosselman. A graph edit dictionary for correcting errors in roof topology graphs reconstructed from point clouds. *ISPRS Journal of Photogrammetry and Remote sensing*, 93, 2014. (Cited on page 109.)

[XSK14] Z. Xu, B.-S. Shin, and R. Klette. Accurate and robust line segment extraction using minimum entropy with hough transform. *Trans. on Image Processing*, 24(3), 2014. (Cited on page 15.)

[XT19] Q. Xu and W. Tao. Multi-scale geometric consistency guided multi-view stereo. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2019. (Cited on page 6.)

[ZBKB08] L. Zebedin, J. Bauer, K. Karner, and H. Bischof. Fusion of feature-and area-based information for urban buildings modeling from aerial imagery. In *Proc. of European Conference on Computer Vision (ECCV)*, 2008. (Cited on page 108.)

[ZCS+10] Z. Zhang, Y. Cao, D. Salvi, K. Oliver, J. Waggoner, and S. Wang. Free-shape subwindow search for object localization. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2010. (Cited on page 12.)

[ZFW+12] Z. Zhang, S. Fidler, J. Waggoner, Y. Cao, S. Dickinson, J. Siskind, and S. Wang. Superedge grouping for object localization by combining appearance and shape informations. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2012. (Cited on page 12.)

[ZJ16] Q. Zhou and A. Jacobson. Thingi10k: A dataset of 10, 000 3d-printing models. *arXiv:1605.04797*, 2016. (Cited on pages 42 and 81.)

[ZN09] Q.-Y. Zhou and U. Neumann. A streaming framework for seamless building reconstruction from large-scale aerial lidar data. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2009. (Cited on page 108.)

[ZN10] Q.Y. Zhou and U. Neumann. 2.5*d* dual contouring: A robust approach to creating building models from aerial lidar point clouds. In *Proc. of European Conference on Computer Vision (ECCV)*, 2010. (Cited on page 108.)

[ZN12] Q.Y. Zhou and U. Neumann. 2.5 D building modeling by discovering global regularities. In *Proc. of Computer Vision and Pattern Recognition (CVPR)*, 2012. (Cited on page 108.)

[ZS01] H. Zho and R. Shibasaki. Reconstructing urban 3d model using vehicle-borne laser range scanners. In *Proc. of International Conference on 3-D Digital Imaging and Modeling*, 2001. (Cited on page 5.)

[ZSG+18] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. State of the art on 3d reconstruction with rgb-d cameras. In *Computer graphics forum*, volume 37, 2018. (Cited on page 7.)

[ZSGH18] L. Zhu, S. Shen, X. Gao, and Z. Hu. Large scale urban scene modeling from mvs meshes. In *Proc. of European Conference on Computer Vision (ECCV)*, 2018. (Cited on page 108.)

[ZWF18] H. Zeng, J. Wu, and Y. Furukawa. Neural procedural reconstruction for residential buildings. In *Proceedings of the Euro-*

pean Conference on Computer Vision (ECCV), 2018. (Cited on page 108.)

[ZZ18] L. Zhang and L. Zhang. Deep learning-based classification and reconstruction of residential scenes from large-scale point clouds. *Trans. on Geoscience and Remote Sensing*, 56(4), 2018. (Cited on page 109.)